

**NEAR EAST UNIVERSITY**



**Faculty of Engineering**

**Department of Computer Engineering**



**DISTRIBUTED BANK SYSTEM**

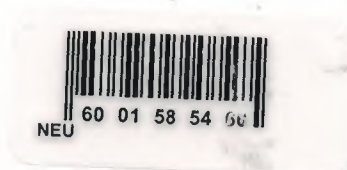
**Graduation Project**

**COM- 400**

**Student: Imad Ahmed Dahdouh**

**Supervisor: Assoc. Prof. Dr. Rahib Abiyev**

**Nicosia – 2003**





## ACKNOWLEDGEMENTS

I would like to thank my supervisor Assist. Prof.Dr Rahib Abiyev for his valuable advice given throughout the duration of this project.

I am also thankful to Prof. Dr. Fakhraddin Mamedov and Assoc.Prof.Dr Adnan Khashman for their support during my studies at Near East University.

And a big thank to my family that supports me in every step I take in this life, and I would like to make use of this chance to thank my mother, father, brothers and my sisters.

As well as I will not forget my friends Reyad Bader, Hany Jaber, Alaa Al-Attar, Nedal Meshal, Rami Al-Dahdouh, Ayman Ashour and Khaled Al Masri who gave me all support during my studying period.

## **ABSTRACT**

Load balancing, administration centralization, platform independency, transactions transparency and object-oriented programming are the main topics in today's system development techniques.

This project discusses the most common issues that related to banking systems. It introduces an easy and efficient way for handling common banking operations such as, money transfer, withdraw, deposit and invoices payment.

A distributed Java-based program has been developed to allow users to deal with the bank system regardless of there places or platforms.

## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b>	<b>1</b>
<b>ABSTRACT</b>	<b>2</b>
<b>TABLE OF CONTENTS</b>	<b>3</b>
<b>TABLE OF FIGURES</b>	<b>6</b>
<b>TABLE OF TABLES</b>	<b>8</b>
<b>INTRODUCTION</b>	<b>9</b>
<b>CHAPTER ONE</b>	
<b>DISTRIBUTED APPLICATIONS ARCHITECTURES</b>	<b>10</b>
1.1 INTRODUCTION	10
1.2 CLIENT/SERVER FUNDAMENTALS	10
1.3 CLIENT/SERVER DISTRIBUTED COMPUTING MODEL	12
1.3.1 Structural Components	13
1.3.2 Functional Components	14
1.4 DISTRIBUTED APPLICATION DEVELOPMENT	15
1.4.1 Distribute Data and Functions	15
1.4.2 Define Communication Protocols	16
1.4.3 Allocate Tasks to Separate Threads	16
1.4.4 Keep Everything Platform and Implementation Independent	16
<b>CHAPTER TWO</b>	
<b>COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)</b>	<b>17</b>
2.1 INFRASTRUCTURE ALTERNATIVES	17
2.2 WHAT IS CORBA?	18
2.3 CORBA ARCHITECTURE	19
2.4 HOW DOES CORBA "GLUE" OBJECTS TOGETHER?	19
2.5 WHAT DOES JAVA OFFER TO CORBA PROGRAMMERS?	20
2.5.1 Portability across platforms	20
2.5.2 Internet programming	20
2.5.3 Object-Oriented language	20
2.5.4 Component model	20
2.6 WHAT DOES CORBA OFFER TO JAVA PROGRAMMERS?	21
2.6.1 Interfaces defined independently of implementations	21
2.6.2 Programming language independence	21
2.6.3 Location transparency and server activation	21
2.6.4 Reuse of CORBA services and facilities	21
2.7 WEB, JAVA AND CORBA	22

2.8 PROBLEMS IN NON-JAVA WEB APPLICATIONS	22
2.9 HOW CORBA SOLVES THE PROBLEM	23
2.10 CORBA & RMI DIFFERENCES	23
<b>CHAPTER THREE</b>	
<b>JAVA DATABASE CONNECTIVITY (JDBC)</b>	<b>24</b>
3.1 INTRODUCTION TO JDBC	24
3.3 JDBC ADVANTAGES	25
3.4 WHAT DOES JAVA PROVIDE?	25
<b>CHAPTER FOUR</b>	
<b>BANK SYSTEM ANALYSIS</b>	<b>27</b>
4.1 SYSTEM GOALS	27
4.2 FEASIBILITY STUDIES	27
4.2.1 Technical Feasibility:	27
4.2.2 Economic Feasibility:	28
4.2.3 OPERATION FEASIBILITY:	29
4.3 DATA DICTIONARY	36
4.4 TABLES DESCRIPTION	37
<b>CHAPTER FIVE</b>	
<b>INPUTS OUTPUTS DESIGN</b>	<b>40</b>
5.1 SEVER INPUTS DESIGNS	40
5.1.1 Server Main Form	40
5.1.2 Server Query Form	41
5.1.3 Server Chart Form	41
5.1.4 Server Reports Form	42
5.1.5 Server E-Mails Form	42
5.1.6 Server Add Employee Form	43
5.2 EMPLOYEE CLIENT INPUTS DESIGNS	44
5.2.1 Employee Client Main Form	44
5.2.2 Employee Client Check Balance Form	45
5.2.3 Employee Client Deposit Form	45
5.2.4 Employee Client Withdraw Form	46
5.2.5 Employee Client Transfer Form	46
5.2.6 Employee Client Services Form	47
5.2.7 Employee Client Reports Form	47
5.2.8 Employee Client New User Form	48
5.2.9 Employee Client Update User Profile Form	48
5.2.10 Employee Client Change User Password Form	49
5.2.11 Employee Client Show Currencies Form	49
5.3 USER CLIENT INPUTS DESIGNS	50
5.3.1 User Client Main Form	50
5.3.2 User Client Check Balance Form	51



5.3.3 User Client Transfer Money Form	51
5.3.4 User Client Services Form	52
5.3.5 User Client Reports Form	52
5.3.6 User Client Update User Profile Form	53
5.3.7 User Client Change User Password Form	53
<b>5.4 SERVER OUTPUTS DESIGNS:</b>	<b>54</b>
5.4.1 Users Details Report	54
5.4.2 Employees Details Report	54
5.4.3 Currencies Prices Report	55
5.4.4 Withdraw Operations Report	55
5.4.5 Deposit Operations Report	56
5.4.6 Export Transfer Operations Report	56
5.4.7 Import Transfer Operations Report	57
5.4.8 Invoices Operations Report	57
<b>5.5 CLIENTS OUTPUTS DESIGNS:</b>	<b>58</b>
5.5.1 User Withdraw Operations Report	58
5.5.2 User Deposit Operations Report	58
5.5.3 User Export Transfer Operations Report	59
5.5.4 User Import Transfer Operations Report	59
5.5.5 User Invoices Operations Report	60
<b>CONCLUSION</b>	<b>61</b>
<b>REFERENCES</b>	<b>62</b>
<b>APPENDIX A</b>	
<b>PROGRAM SOURCE CODE</b>	<b>63</b>

## TABLE OF FIGURES

Figure 1.1 Conceptual Client/Server Model	11
Figure 1.2 Interrelationships between Computing Models	12
Figure 1.3: Distributed Application Structure Flowchart	13
Figure 1.4: Distributed Application Development	15
Figure 4.1: Client login operation	29
Figure 4.2: Server login operation	30
Figure 4.3: Deposit operation	31
Figure 4.4: Withdraw Operation	31
Figure 4.5: Transfer operation	32
Figure 4.6: Check balance operation	32
Figure 4.7: Change password	33
Figure 4.8: Add new user or employee operation	33
Figure 4.9: Drop user operation	34
Figure 4.10: Drop employee operation	34
Figure 4.11: Context Diagram	35
Figure 4.12: Entity Relationships Model	36
Figure 5.1: Server Main Form	40
Figure 5.2: Server Query Form	41
Figure 5.3: Server Chart Form	41
Figure 5.4: Server Reports Form	42
Figure 5.5: Server E-Mails Form	42
Figure 5.6: Server Add Employee Form	43
Figure 5.7: Employee Client Main Form	44
Figure 5.8: Employee Client Check Balance Form	45
Figure 5.9: Employee Client Deposit Form	45
Figure 5.10: Employee Client Withdraw Form	46
Figure 5.11: Employee Client Transfer Form	46
Figure 5.12: Employee Client Services Form	47
Figure 5.13: Employee Client Reports Form	47
Figure 5.14: Employee Client New User Form	48
Figure 5.15: Employee Client Update User Profile Form	48
Figure 5.16: Employee Client Change User Password Form	49
Figure 5.17: Employee Client Show Currencies Form	49
Figure 5.18: User Client Main Form	50
Figure 5.19: User Client Check Balance Form	51
Figure 5.20: User Client Transfer Money Form	51
Figure 5.21: User Client Services Form	52
Figure 5.22: User Client Reports Form	52
Figure 5.23: User Client Update User Profile Form	53
Figure 5.24: User Client Change User Password Form	53
Figure 5.25: Users Details Report	54
Figure 5.26: Employees Details Report	54
Figure 5.27: Currencies Prices Report	55
Figure 5.28: Withdraw Operations Report	55

Figure 5.29: Deposit Operations Report	56
Figure 5.30: Export Transfer Operations Report	56
Figure 5.31: Import Transfer Operations Report	57
Figure 5.32: Invoices Operations Report	57
Figure 5.33: User Withdraw Operations Report	58
Figure 5.34: User Deposit Operations Report	58
Figure 5.35: User Export Transfer Operations Report	59
Figure 5.36: User Import Transfer Operations Report	59
Figure 5.37: User Invoices Operations Report	60



## TABLE OF TABLES

Table 2.1: RMI & CORBA Differences	23
Table 4.1: Deposit Database Table	37
Table 4.2: EmpProfile Table	37
Table 4.3: Emps Table	37
Table 4.4: Hosts Table	37
Table 4.5: Prices Table	38
Table 4.6: Profile Table	38
Table 4.7: Services Table	38
Table 4.8: Transfer Table	38
Table 4.8: Users Table	39
Table 4.10: Withdraw Table	39

## INTRODUCTION

Every day there are millions or billions of money transformation transactions occurs all over the world. Handling this amount of money deals with the traditional manual techniques will really be a useless method. With today's new technologies and global free services the work goes much easier, cheaper and faster.

The banking system that we are introducing, gives both dealers and bank employees an optimal solution to finish there jobs more efficient with a least efforts. Using Java as my programming language makes this project a real scalable and platform-independent solution for our days sophisticated needs. Many challenges have been raised because of system distribution, like security and data integrity. Through the project we will find solutions for those and other critical points.

With a lot of alternatives, we could say that choosing distributed programming and Java Infrastructure give us the best results and solutions. Other languages like C++, Delphi or Pascal lacks from network programming support, other that, they are all platform-dependent. With those languages the solution would be narrow and weak one. And it is true that network is an insecure environment, related to stand-alone systems, for bank critical transactions, but choosing and developing the correct security techniques will absolutely overcome this challenge.

# CHAPTER ONE

## DISTRIBUTED APPLICATIONS ARCHITECTURES

### 1.1 Introduction

An object-oriented client/server Internet (OCSI) environment provides the IT an infrastructure (i.e., middleware, networks, operating systems, hardware) that supports the OCSI applications. The purpose of this chapter is to explore this enabling infrastructure before digging deeply into the details of our banking system design. Specifically, we review the following three core technologies of the modern IT infrastructures:

- Client/server that allows application components to behave as service consumers (clients) and service providers (servers).
- Internet for access to application components (e.g., databases, business logic) located around the world from Web browsers.
- Object-orientation to let applications behave as objects that can be easily created, viewed, used, modified, reused, and deleted over time.

### 1.2 Client/Server Fundamentals

Client/server model is a concept for describing communications between computing processes that are classified as service consumers (clients) and service providers (servers).

Figure 1.1 presents a simple C/S model. The basic features of a C/S model are:

1. Clients and servers are functional modules with well defined interfaces (i.e., they hide internal information). The functions performed by a client and a server can be implemented by a set of software modules, hardware components, or a combination thereof. Clients and/or servers may run on dedicated machines, if needed. It is unfortunate that some machines are called "Servers". This causes some confusion, but bewildered users know that client soft wares are also running on a server-called machine.
2. Each client/server connection is established between two functional modules when one module (client) initiates a service request and the other (server) chooses to respond to the service request.

3. Information exchange between clients and servers is strictly through messages (i.e., no information is exchanged through global variables). The service request and additional information is placed into a message that is sent to the server. The server's response is similarly another message that is sent back to the client. This is an extremely crucial feature of C/S model.
4. Messages exchanged are typically interactive. In other words, C/S model does not support an off-line process. There are a few exceptions. For example, message queuing systems allow clients to store messages on a queue to be picked up asynchronously by the servers at a later stage.
5. Clients and servers typically reside on separate machines connected through a network. Conceptually, clients and servers may run on the same machine or on separate machines. However, our primary interest is in *distributed client/server systems* where clients and servers reside on separate machines.

The implication of the last two features is that C/S service requests are real-time messages that are exchanged through network services. This feature increases the appeal of the C/S model (i.e., flexibility, scalability) but introduces several technical issues such as portability, interoperability, security, and performance.

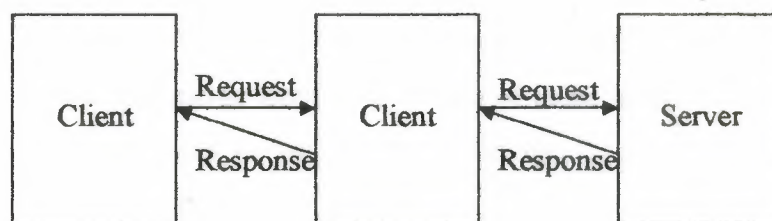


Figure 1.1 Conceptual Client/Server Model



### 1.3 Client/Server Distributed Computing Model

Figure 1.2 shows the interrelationships between distributed computing and client/server models. Conceptually, client/server model is a special case of distributed-computing model.

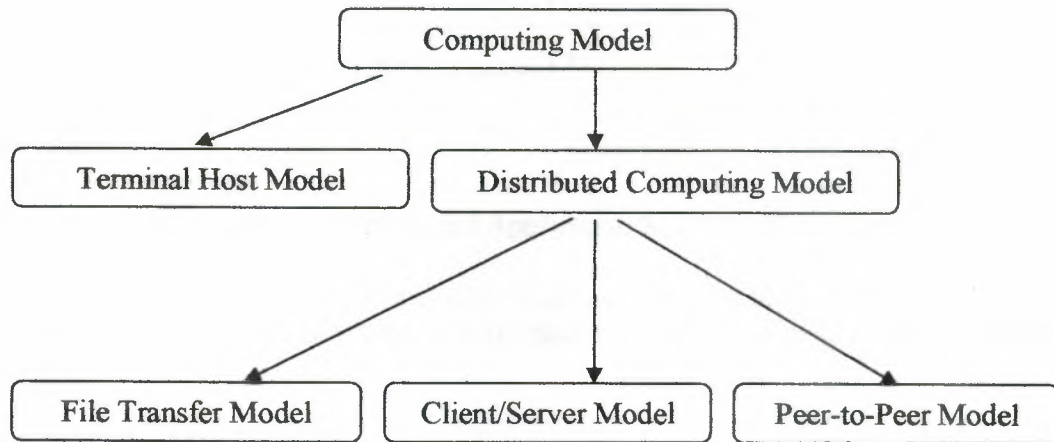


Figure 1.2 Interrelationships between Computing Models

Distributed Computing System (DCS) is a collection of autonomous computers interconnected through a communication network to achieve business functions. A distributed application is built upon several layers. At the lowest level, a network connects group of host computers together so that they can talk to each other. Network protocols like TCP/ IP let the computers send data to each other over the network by providing the ability to package and address data for delivery to another machine. Higher-level services can be defined on top of the network protocol, such as directory services and security protocols. Finally the distributed application itself runs on top of these layers, using the mid-level services and network protocols as well as the computer operating systems to perform coordinated tasks across the network.



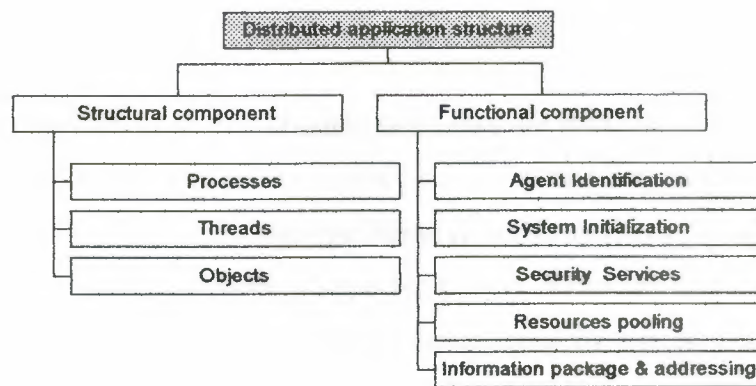


Figure 1.3: Distributed Application Structure Flowchart

Distributed application requires the structural components and functional components as shown in figure 1.3. These structural and functional components are described in detail below.

### 1.3.1 Structural Components

Distributed application can be broken down into the following structural components:

#### 1. Processes

A typical computer operating system on a computer host can run several processes at once. A process is created by describing a sequence of steps in a programming language, compiling the program into an executable form, and running the in the operating system. While it's running, a process has access to the resources executable of the computer (such as CPU time and I/ O devices) through the operating system. A process can be completely devoted to a particular application, or several applications can use a single process to perform tasks.

#### 2. Threads

Every process has at least one thread of control. Some operating systems support the creation of multiple threads of control within a single process. Each thread in a process can run independently from the other threads, although there is usually some synchronization between them. One thread might monitor input from a socket connection, for example, while another might listen for user events (keystrokes, mouse movements, etc.) and provide feedback to the user through output devices (monitor, speakers, etc.). At some point, input from the input stream may require feedback from the user. At this point, the two threads will need to coordinate the transfer of input data to the user's attention.

### **3. Objects**

Programs written in object- oriented languages are made up of cooperating objects. One simple definition of an object is a group of related data, with methods available for querying or altering the data (`getName ()`, `setName ()`), or for taking some action based on the data (`sendName (OutputStream o)`). A process can be made up of one or more objects, and these objects can be accessed by one or more threads within the process.

#### **1.3.2 Functional Components**

In addition to the structural components described above, nearly all distributed Require the following functional components.

##### **1. Information Packaging, Addressing and Delivery**

Since the components of a distributed system are, by definition, distributed, there needs to be a way to package up information, address it correctly.

##### **2. Agent Identification**

Implied by the need for information addressing is the need to be able to explicitly identify the agents in a distributed system. These agents might be software entities (groups of objects running on an application server), or human beings (buyers submitting bids in an online auctioning system). Just as servers need to address data packets sent to each other at the wire protocol level using IP addresses, agents at the application level need to address messages that they send to each other using some kind of identification scheme.

##### **3. System Initialization**

In any software system, there is some kind of initialization process that takes place when the system first comes to life. Distributed applications first, make some initialization procedures before become alive. Sometimes this can be as simple as starting a distributed server process and letting it wait for clients to connect (e. g., a basic HTTP server).

##### **4. Security Services**

If you're spreading out the components of a system across remote hosts, and these hosts are talking to each other over network connections, then to some degree there

needs to be some consideration for security services. Sensitive data sent over network connections may need to be secured from prying “eyes” tapping the wires.

## 5. Resource Pooling and Transaction Services

A distributed system will, by definition, involve more than one computing entity (process, thread, object, agent) interacting over the network. If many entities need access to the same resource to do what they need to do, then that resource may need to be pooled in order to improve the lag time in servicing these agents, and the resource may need to be wrapped with transaction services to keep its state consistent. The agents might be software entities (groups of objects running on an application server).

### 1.4 Distributed Application Development

Now that we’ve defined some of the building blocks, these are some of the typical steps that go into developing distributed systems as shown in figure 1.4. Some kinds of tools and capabilities that you’ll need in order to take these steps are also mentioned. The following is simply an overview of these topics.

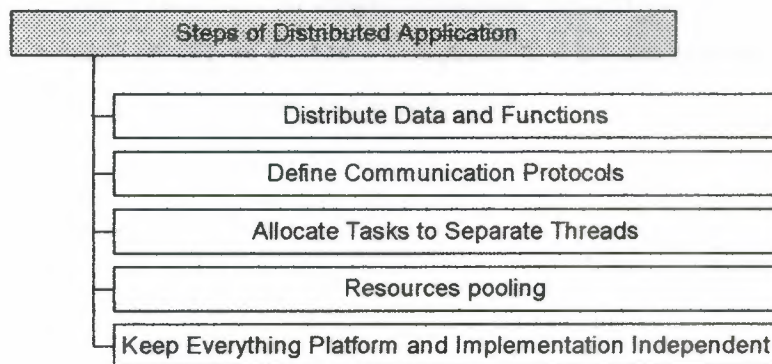


Figure 1.4: Distributed Application Development

#### 1.4.1 Distribute Data and Functions

If you think that hosts and network connections are all available for a distributed application to use as a “virtual machine”, then one of the primary tasks you have is to engineer an optimal mapping of processes, objects and threads to the various parts of this virtual machine. Computational tasks can be distributed based on the data needs of the application: maximize local data needed for processing, and minimize data transfers over the network.



application: maximize local data needed for processing, and minimize data transfers over the network.

#### **1.4.2 Define Communication Protocols**

The type and format of the information that's sent between agents in a distributed system is a subject to many changing requirements. We know that there are two common kinds of communication protocols, TCP/IP and UDP protocols. According to the application needs we can decide which protocol to use.

#### **1.4.3 Allocate Tasks to Separate Threads**

Server often has to execute several threads of control and other threads to service requests from multiple remote clients. Multithreading is often an effective way to optimize the use of various resources, such as CPU time, local storage devices, or network bandwidth. The ability to create and control multiple threads of control is especially important in developing distributed applications.

#### **1.4.4 Keep Everything Platform and Implementation Independent**

Any distributed system should be platform-independent. Through platform independency, distributed systems will be expandable and widely used.

## **CHAPTER TWO**

### **COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)**

#### **2.1 Infrastructure Alternatives**

Tools and standards for distributed applications have been developed over the years. Some of these standers are:

- Peer-to-Peer.
- RMI (Remote Method Invocation)
- CORBA (Common Object Request Broker Architecture).
- And others.

In our project we've used CORBA for solving our application problem. We will explain below why CORBA's been chosen among other infrastructures.

As a first alternative, we can solve our problem using sockets, but it will be very difficult and complex. Developing our solution using sockets requires building the solutions from the lowest sockets layer to the highest user interface layer, which will of course consumes and duplicates our development time and efforts.

In the other hand the problem can be solved by servlets, but we prefer CORBA and RMI upon it, because in CORBA and RMI we can write every thing by our selves but servlet restricts us to use certain style of applications. But the main advantage of servlets is that we can easily build a client and run it on the browser.

We can not build our application by using Peer-to-Peer infrastructure because the term peer- to- peer refers to a distributed system where the various agents are roughly on equal footing. There's no obvious server or client, everyone can talk to each other and pass data between each other which violates our solution security system requirements.



## 2.2 What is CORBA?

The Common Object Request Broker Architecture (CORBA) is the most important middleware project ever undertaken by industry. It is the product of a consortium called OMG (Object Management Group) that includes over 700 companies, representing an entire spectrum of the modern computer industry. Microsoft, which has its own competing product broker called Distributed Component Object Model (DCOM) is not part of this group.

CORBA is an open standard for building distributed objects that can interoperate with each other. Traditional Object-Oriented programming restricts an object to communicate with other objects on the same machine. CORBA has the power to extend the address space of a program to the entire network.

CORBA is designed to allow intelligent components to discover each other and interoperate on an object bus. CORBA goes beyond just simple interoperability; it also specifies an extensive set of bus-related services for creating and deleting objects, accessing them by name, storing them in persistent stores, externalizing their states, and defining ad hoc relationships between them.

What makes CORBA so important is that it defines a middleware and has the potential to include any other form of existing client/server middleware. In other words, CORBA uses objects as a unified approach for bringing existing applications to the bus. It provides a solid foundation for a component-based future. The power of CORBA is that the entire system is self-describing. The specification of a service is always separated from the implementation. This gives the users the possibility of incorporating existing systems within the bus.

CORBA makes it possible to create an ordinary object and then make it transactional, secure, lockable, and persistent by making the object able to inherit needed services from the appropriated servers. This means that one can design an ordinary component to provide its regular function, and then insert the right middleware mix when

you build it or create it in run time. There is nothing like it provided by any existing client/server environments.

### **2.3 CORBA Architecture**

In order to understand CORBA's components, let's consider the VisiBroker (one of the commercial CORBA vendors) implementation of the CORBA architecture. The essential parts to understand CORBA are:

- ORB (Object Request Broker). This is the CORBA bus that provides a variety of services that allow client objects to transparently make requests to and receive responses from server objects located locally or remotely. The ORB, which is the heart of CORBA, is responsible for:
  - 1) Finding the object implementation for the request.
  - 2) Preparing the object implementation for the request.
  - 3) Communicating the data making up the request.
- IIOP (Internet Inter-Orb Protocol) is a set of standards for ORB-to-ORB communications. In other words, IIOP is the "wire protocol" that defines how messages are sent from client to server across the network.
- BOA (Basic Object Adapter) is the mechanism that activates the server objects so they can receive requests from clients. It is BOA's job to deactivate these objects when they are no longer used.
- Stub and Skeleton classes are automatically generated based on created server objects. These classes handle the marshaling and unmarshaling of parameters. The stub is sent to the client applet and the skeleton remains on the server to interface to object implementation. These classes handle all networking for the client and server objects making the network transparent to the programmer.

### **2.4 How does CORBA "glue" objects together?**

CORBA is not a programming language, it is an integration technology. CORBA uses IDL (Interface Definition Language) to define object's public interface so other objects can use them to communicate with the object.

IDL is purely a declarative language that does not have any implementation. IDL defines the types of objects by defining their interfaces. An interface consists of a set of named operations and the parameters to those operations. It is through IDL that a particular object implementation tells its potential clients what operations are available and how they should be invoked.

From the IDL definitions, the CORBA objects are mapped into different languages. Some of the languages that have IDL mappings are: Java, C++, C, Smalltalk, Add etc.

## **2.5 What does Java offer to CORBA programmers?**

### **2.5.1 Portability across platforms**

Java programs are highly portable due to the standardized byte-code generated by Java compilers. Industry is providing compilers and run-time systems for virtually any platform and operating system.

### **2.5.2 Internet programming**

Java language binding allows implementation of CORBA clients as applets. This allows access to legacy data using popular browsers.

### **2.5.3 Object-Oriented language**

Java ORB's provide the same functionality as any other ORB. Java provides a cleaner approach to object-oriented programming than C++. Additionally, Java provides features not available in C++ or C, such as garbage collection.

### **2.5.4 Component model**

Java beans are the most recent addition to the core of Java programming language. The component model allows programmers to combine the functionality provided by many Java classes into a single component. Components can be easily put together to achieve new functionality.



## **2.6 What does CORBA offer to Java programmers?**

### **2.6.1 Interfaces defined independently of implementations**

OMG IDL provides a means of separating interfaces from implementations for distributed objects applications. Once interfaces are defined different teams can implement them separately.

### **2.6.2 Programming language independence**

CORBA supports multiple language mappings for OMG IDL so different part of the system can be implemented in different languages. All interactions in the system happen through interfaces that are specified independently of any language.

### **2.6.3 Location transparency and server activation**

Socket or URL based distributed applications need to address a server by specifying a host name or a port number. CORBA provides location transparency; an object is identified independently of its physical location without breaking the application. The ORB provides the mechanisms for this transparency. CORBA provides mechanisms to start up services on demand that can be controlled by various server activation policies.

### **2.6.4 Reuse of CORBA services and facilities**

The ORB provides means for the distribution-transparent invocation of methods on potentially remote objects. Nontrivial distributed applications require additional functionality. Within the OMG these requirements have been analyzed and have led to the specification of fundamental services. These fundamental services are:

- Naming Services
- Trading Services
- Event Services
- Transaction Services
- Security Services

## **2.7 Web, Java and CORBA**

The progression of Web functionality from simple document fetching to more and more complex and interactive applications has followed the following steps:

- Fetching HTML or other formatted documents from fixed locations.
- Fetching documents from back-end systems, such as databases, using the CGI
- Building interactive systems using HTML forms and CGI
- Using Java scripts to increase GUI capabilities
- Using Java applets to provide client-side functionality.

## **2.8 Problems in non-Java web applications**

- Tools provided by HTML to create GUI-s are not sufficient for commercial applications when compared to Windows or Macintosh OS operating systems.
- The interactivity of HTML applications is provided through CGI interfaces or similar functionality provided by certain web server products. These have the following problems:
  1. Clients are stateless, they do not have history. The client is a sequence of HTML pages where each is created as the result of a CGI call. Hence all client state information has to be passed to a program behind the CGI. The only way to do this is by encoding it into the URL.
  2. Writing a client as a sequence of HTML pages and URL is an extremely tedious task and as such, has the potential for many errors. Data transferred from the client to the server must be encoded in the URL string that must be parsed each time a new CGI call is received.
  3. There are a number of bottlenecks in the CGI-based approach. As a result of an invocation a complete HTML page is returned that contains a lot of repeated information. The amount of repeated information outweighs the amount of data produced by the application by an order of magnitude.
- HTTP is not very efficient. The major performance bottleneck occurs because multiple connections can be created by loading a single URL and the connection management creates a significant performance overhead. Furthermore, the CGI will



start a new operating system process each time an application processes a user input.

## 2.9 How CORBA solves the problem

1. Java ORB's overcome the stateless problem by having continuously executing client and server programs which maintain their own state variables.
2. ORB infrastructure allows the convocation of operations on remote objects, which communicate only the data they need for each interaction. The ORB maintains a network connection between client and server, keeping a reasonable trade-off between lowering connection establishment overhead and freeing idle network resources.
3. CORBA follows object-oriented design conventions.

## 2.10 CORBA & RMI Differences

Table 2.1: RMI & CORBA Differences

	RMI	CORBA
Languages supported	Java	Java, C++, C, Smalltalk, etc.
Runtime services	Naming	Naming, Lifecycle, Persistence, Transactions, etc.
Ease of programming and setup	Excellent	Good
Scalability	Good	Excellent (depending on ORB vendor)
Performance	Good	Excellent (depending on ORB vendor)

## **CHAPTER THREE**

### **JAVA DATABASE CONNECTIVITY (JDBC)**

#### **3.1 Introduction to JDBC**

Java Database Connectivity (JDBC) is the industry standard for database-independent connectivity between Java applets/applications and a broad range of SQL databases. All the benefits of "Write Once, Run Anywhere" equally apply to JDBC. The JDBC API defines Java classes that represent database connections, SQL statements, result sets, database metadata, etc.

It allows a Java programmer to do three things:

1. Establish a connection to a database.
2. Issue SQL statements.
3. Process the results.

The JDBC API is implemented via a driver manager that can support multiple drivers connecting to different databases. JDBC drivers can either be entirely written in Java so that they can be downloaded as part of an applet, or they can be implemented using native methods to bridge to existing database access libraries.

#### **3.2 JDBC Architecture**

Applications and Applets may access databases via JDBC using pure Java drivers as follows:

1. **Direct-to Database Pure Java Driver:** This type of driver converts JDBC calls into the network protocol used directly by DBMS's, allowing a direct call from the client machine to the DBMS server and providing a practical solution for intranet access.
2. **Pure Java Driver for Database Middleware:** This type of driver translates JDBC calls into the middleware vendor's protocol, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases. You may also use ODBC drivers and existing database client libraries as part of a JDBC connectivity solution.

3. **JDBC-ODBC Bridge plus ODBC Driver:** The Sun bridge product provides JDBC access via ODBC drivers. Both ODBC binary code and sometimes the database client code must be loaded on each machine that uses this driver.
4. **Native-API Partly Java Driver:** This style of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix and the like. Requires that some binary code be loaded on each client machine.

### **3.3 JDBC Advantages**

1. Businesses can leverage existing enterprise data with JDBC by continuing to use their installed databases and access information easily-even if it's stored on different database management systems.
2. Businesses benefit from reduced development time. The combination of Java and JDBC makes application development easy and economical. JDBC is simple to learn, easy to deploy and inexpensive to maintain.
3. With JDBC, there are zero configurations for network computers because configuration is required on the client side, since the connection is completely defined by the JDBC URL. This supports the network-computing paradigm and centralizes software maintenance.

### **3.4 What Does Java Provide?**

Java is developed at potential Sun Microsystems, with improved supporting for networking, security, and multithreaded operations. All of these features of the Java language and environment incorporated also made for a very powerful distributed application development environment.

Java has properties that make it powerful over other languages as:

- Object- Oriented Environment.
- Network Support.
- Abstract Interfaces.
- Platform Independence.
- Security.
- Multithreading Support.

- It is simple.
- Has lots of powerful stander library.

As we show from the properties, Java gives us the all requirement to build distributed system applications. Because of that we use java language to build our application.



## **CHAPTER FOUR**

### **BANK SYSTEM ANALYSIS**

#### **4.1 System goals**

1. Facilitate operations of deposit, withdraw, transfer and invoices payment.
2. Facilitate statistical operations. For example, an annual or monthly stocking.
3. Facilitate an operation of reviewing and dragging calculations of the bank.
4. Facilitate the operations of obtaining detailed reports in any time.
5. Saving times, this could be wasted in blare operations.
6. Increasing speedy of operations and its interaction.
7. Reserving information's from loss and corruption.
8. Improving security level.
9. Having an ability to do economic studies easily on existing information quickly, to help in developing the system.
10. Getting most of Client/server model that allows application components to behave as service consumers (Clients) and service providers (servers).
11. Cycling the work and create a new system not to cause boredom and routine.

#### **4.2 Feasibility Studies**

##### **4.2.1 Technical Feasibility:**

The system needs several equipments which could be described as follows:

##### **4.2.1.1 Personal computers with the following specifications:**

1. Pentium II or higher.
2. Free disk space 30 MB or bigger.
3. 16 MB for main memory or higher.
4. VGA Monitor.
5. Sound card & Display card.
6. Standard mouse and keyboard.
7. Fax modems (56KB) or Network card.



#### **4.2.1.2 Server computer with the following specifications:**

1. Pentium III or higher.
2. Free disk space 50 MB or bigger.
3. 64 MB for main memory or higher.
4. VGA Monitor.
5. Sound card & Display card.
6. Standard mouse and keyboard.
7. Fax modems (56KB) or Network card.

#### **4.2.1.3 Network equipments:**

1. Hub.
2. UTP Cables.

#### **4.2.1.4 Auxiliary equipments:**

1. Printer like (*Color Printer*).

#### **4.2.2 Economic Feasibility:**

- ≈ The new system decreased number of employers required.
- ≈ The new system decreased time of operations, by maximizing the utilization the bank system.
- ≈ The new system needs some computers, network equipments and starting system like (Windows 95, 98, 2000Pro for Clients and Windows NT, Windows 2000 Server for Server machine).

#### 4.2.3 OPERATION FEASIBILITY:

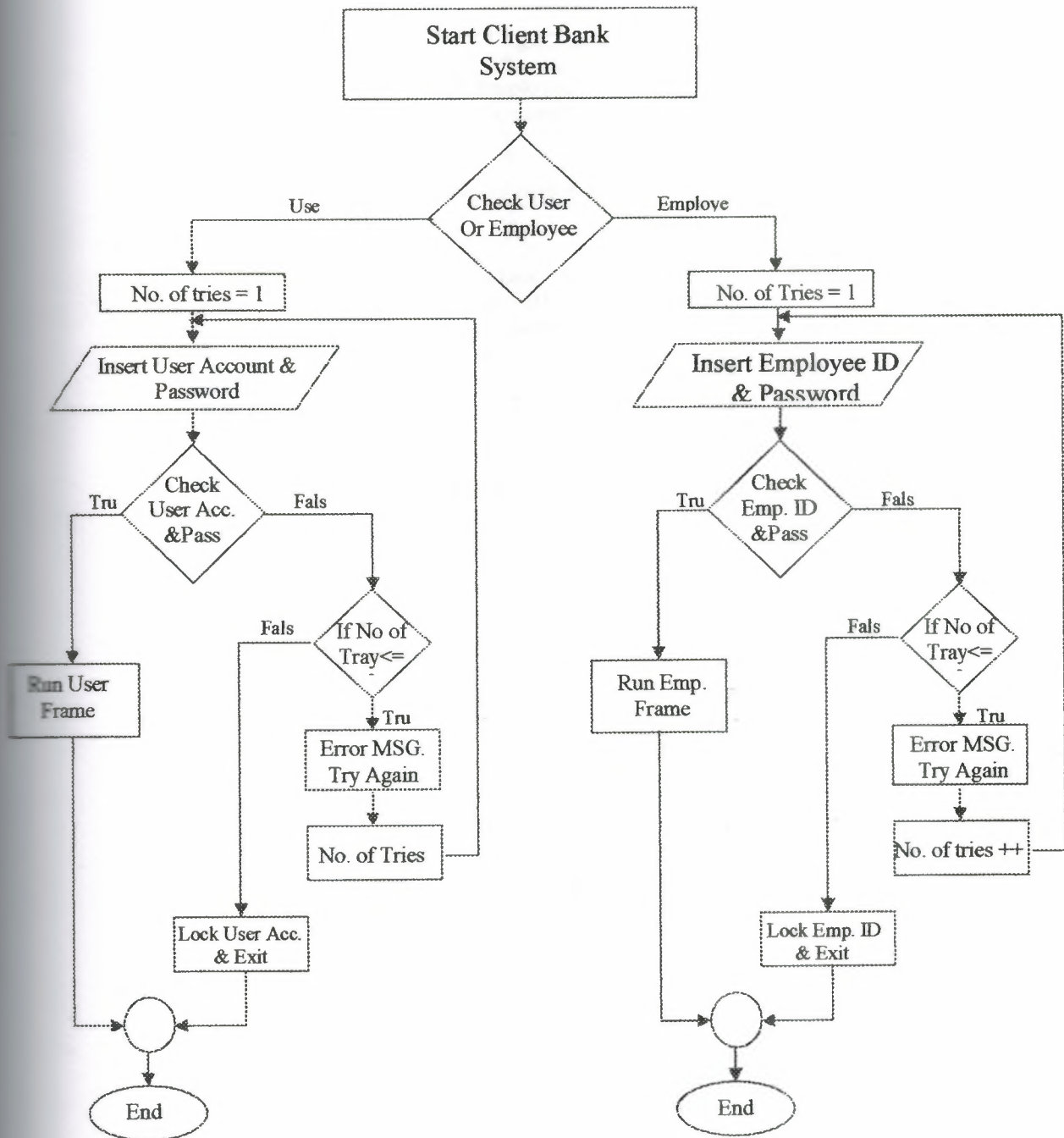


Figure 4.1: Client login operation

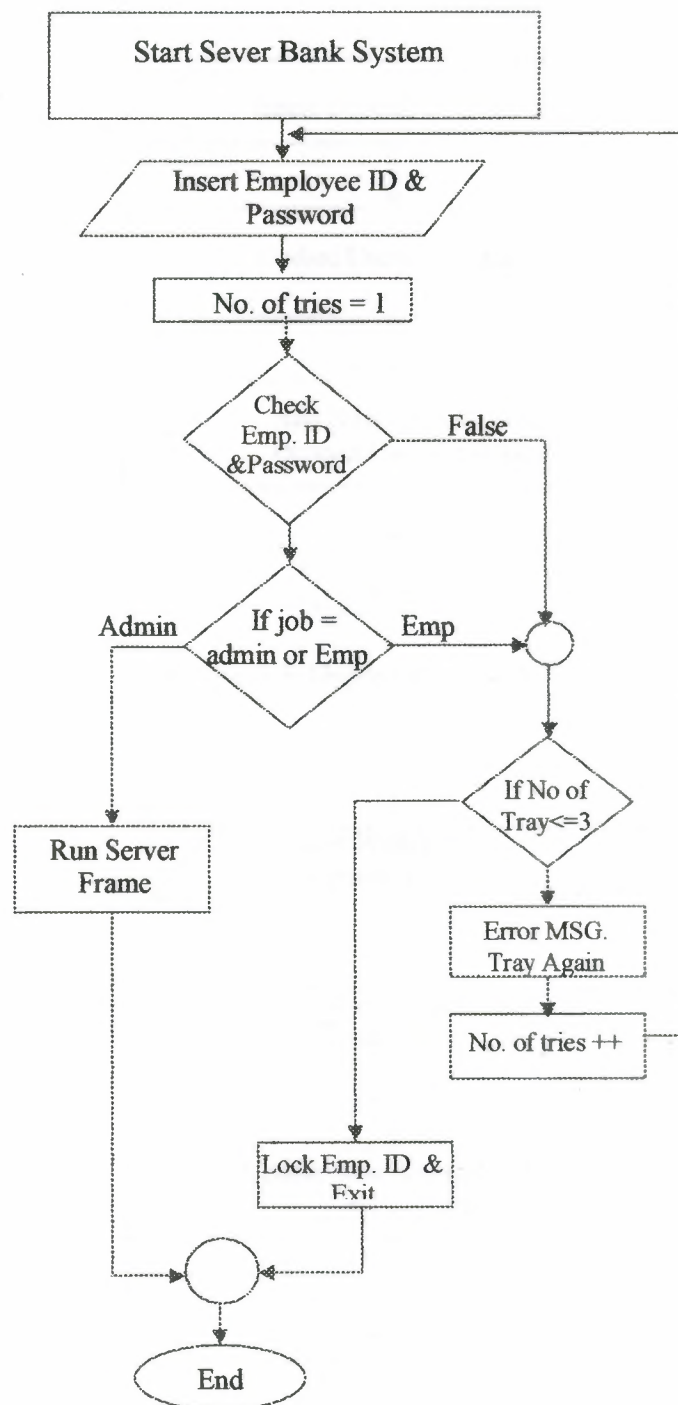


Figure 4.2: Server login operation

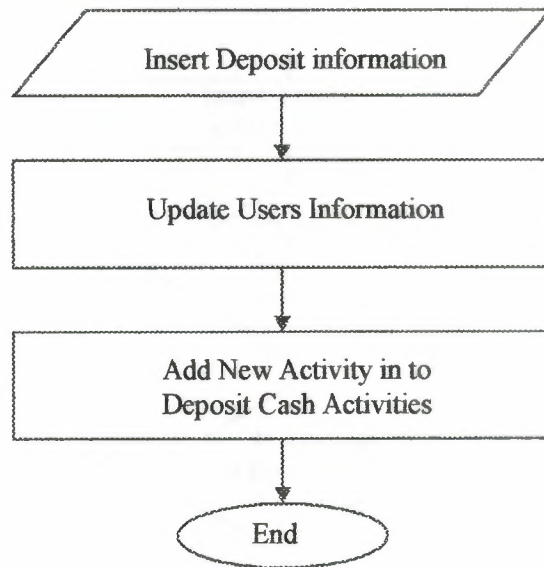


Figure 4.3: Deposit operation

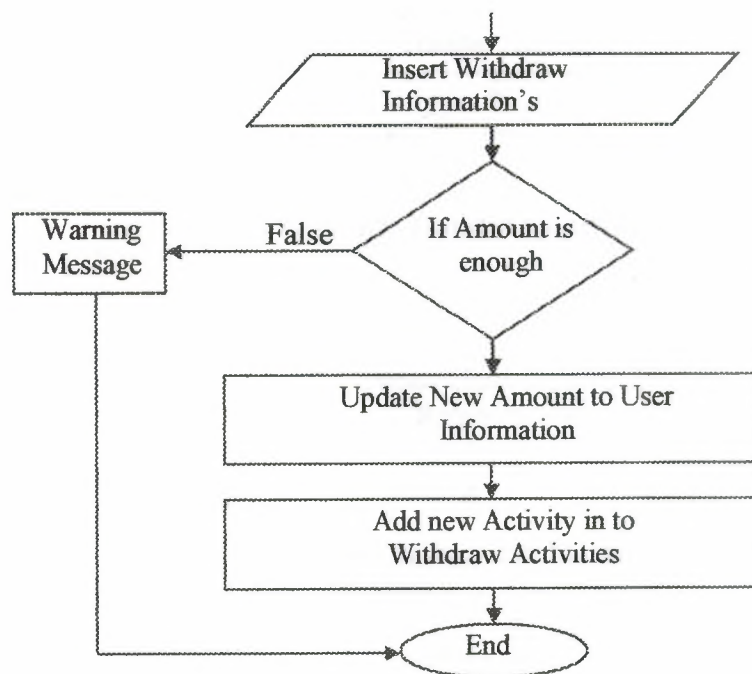


Figure 4.4: Withdraw Operation



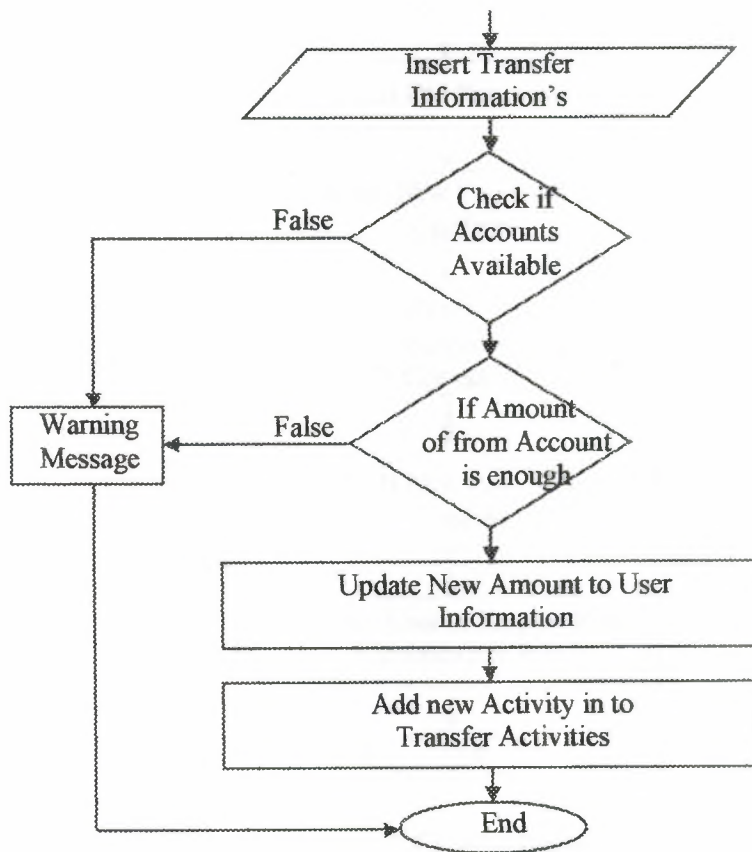


Figure 4.5: Transfer operation

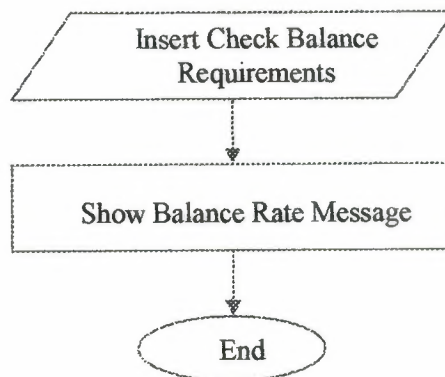


Figure 4.6: Check balance operation

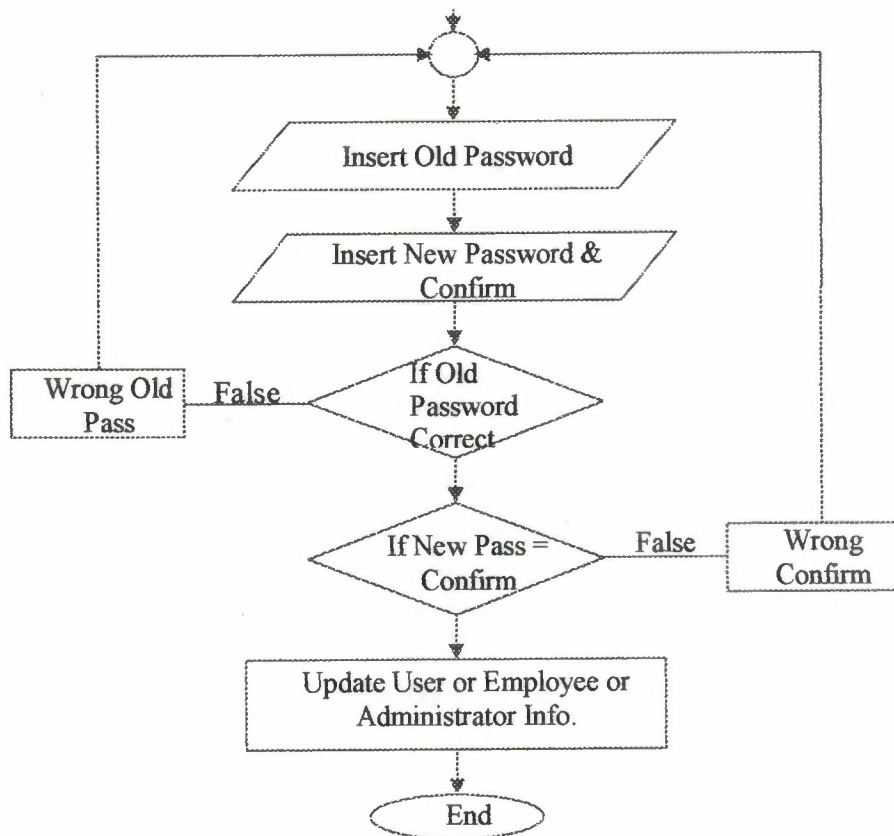


Figure 4.7: Change password

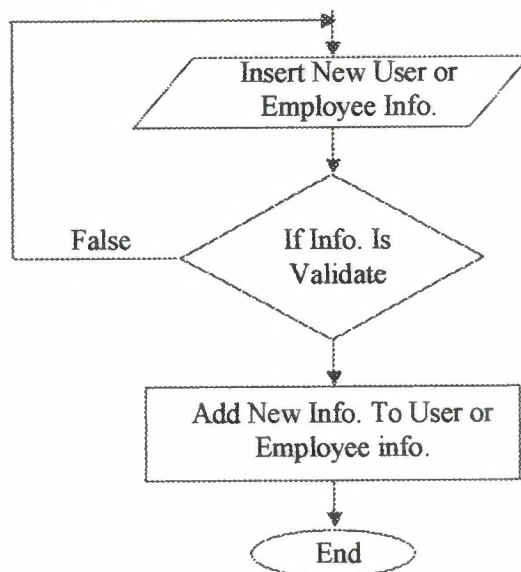


Figure 4.8: Add new user or employee operation

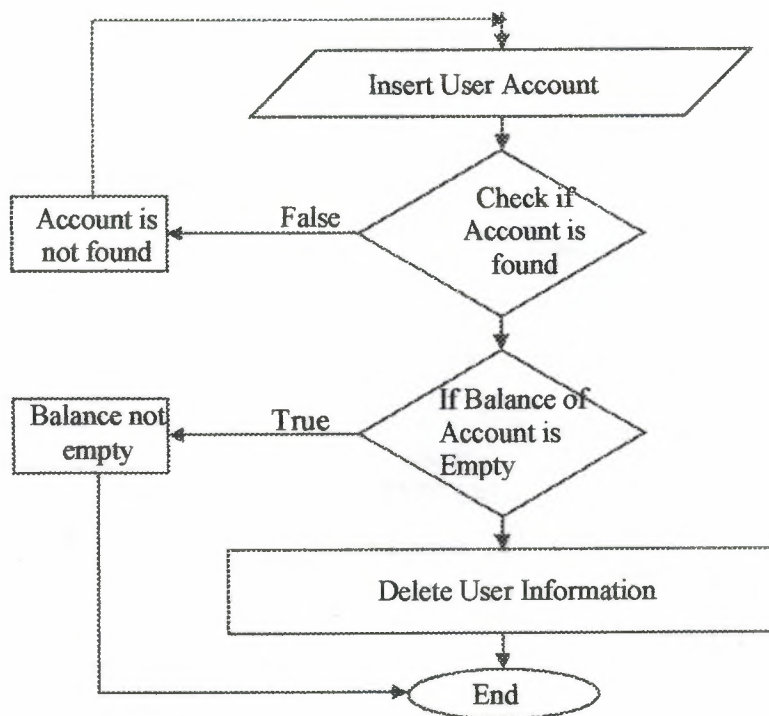


Figure 4.9: Drop user operation

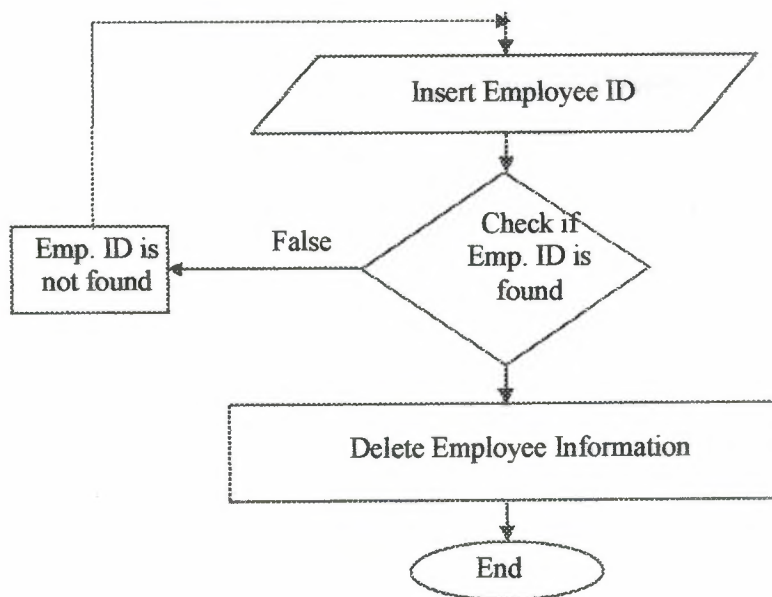


Figure 4.10: Drop employee operation

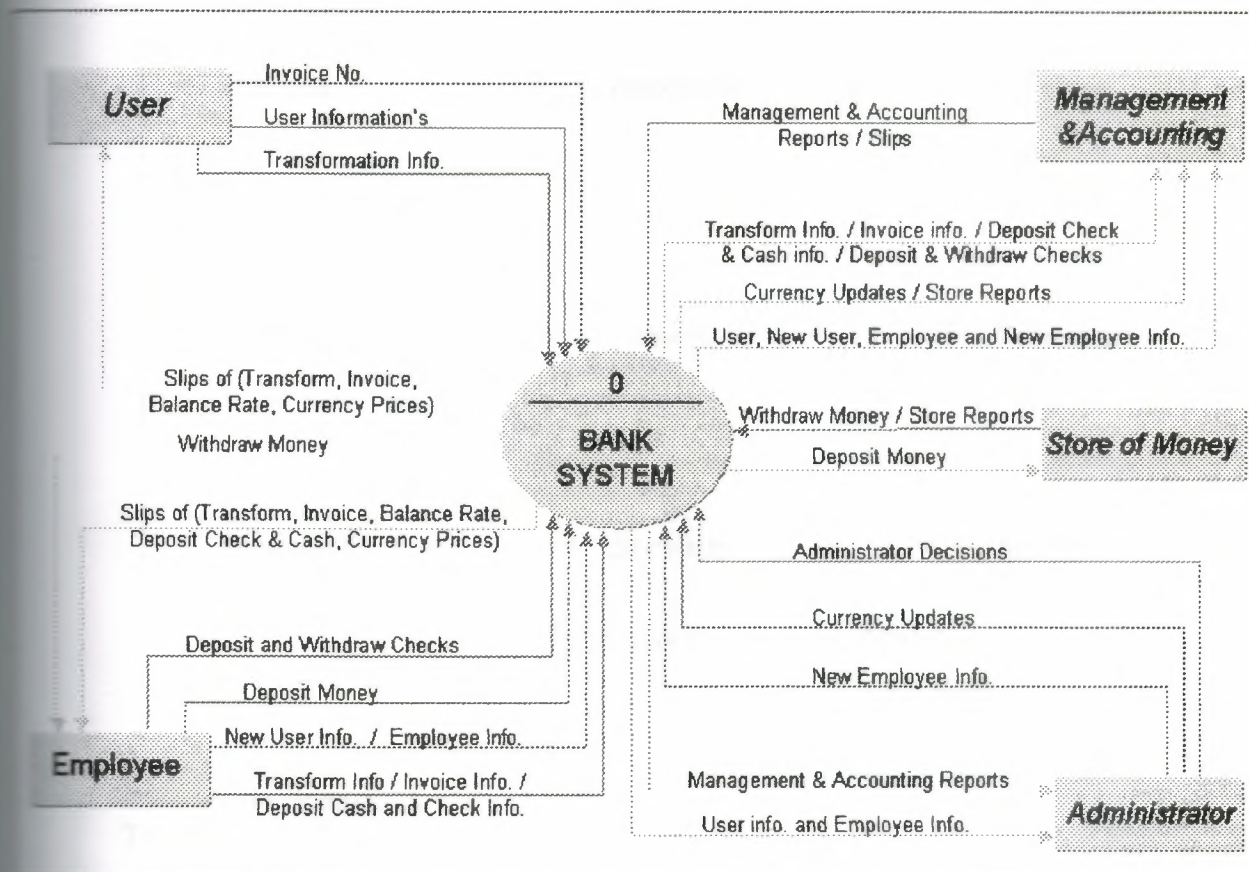


Figure 4.11: Context Diagram



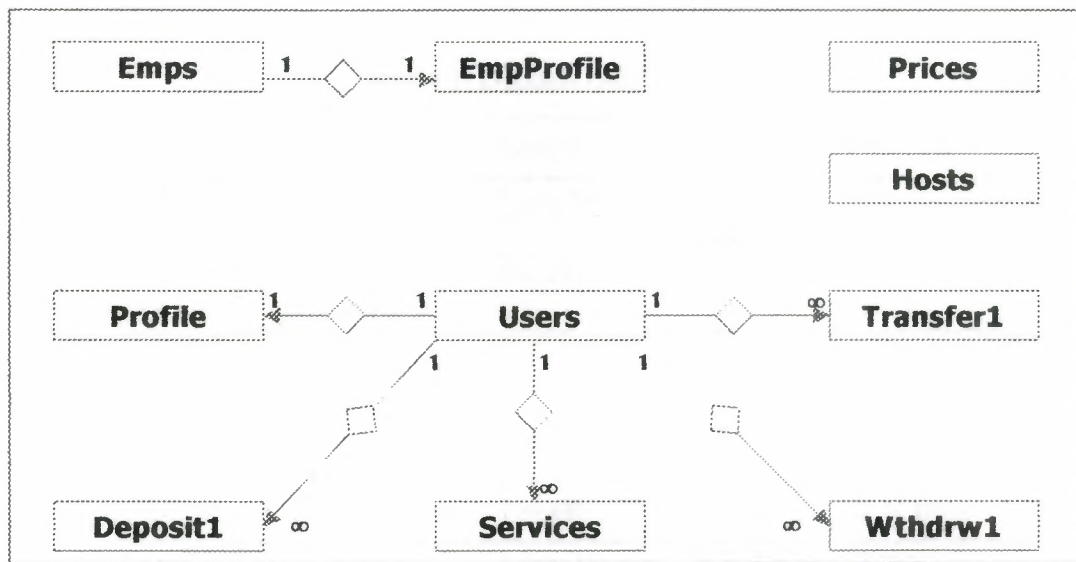


Figure 4.12: Entity Relationships Model

### 4.3 Data Dictionary

Tables:

- **Deposit:** (DID, account\_no, amount, pdate).
- **EmpProfile:** (EmpID, Empemail, SMTPHost, SteEmpaddress, EmpHTele, EmpJTele).
- **Emps:** (EmpID, EmpName, EmpJob, EmpPass, EmpLock).
- **Hosts:** (CompName, Host).
- **Prices:** (curname, currprice).
- **Profile:** (account\_no, email, SMTPHost, address, HTele, JTele, Company).
- **Services:** (invoiceno, invoicetype, account\_no, amount, payeddate).
- **Transfer:** (TID, Facnt\_no, Tacnt\_no, amount, pdate).
- **Users:** (account\_no, name, password, amount, lock, created\_date).
- **Withdraw:** (WID, account\_no, amount, pdate).

#### 4.4 Tables Description

Table 4.1: Deposit Database Table

Field Name	Data Type	Length	Index	F.K Base T.
<u>DID</u>	Number	Long Integer	P.K.	
<u>account_no</u>	Number	Long Integer	P.K.	
Amount	Number	Double		
pdate	Text	30		

Table 4.2: EmpProfile Table

Field Name	Data Type	Length	Index	F.K Base
EmpID	Number	Long Integer	P.K.&F.K.	Emps
Empemail	Text	50		
SMTPHost	Text	50		
Empaddress	Text	50		
EmpHTele	Text	20		
EmpJTele	Text	20		

Table 4.3: Emps Table

Field Name	Data Type	Length	Index	F.K Base
EmpID	Number	Long Integer	P.k	
EmpName	Text	50		
EmpJob	Text	20		
EmpPass	Text	20		
EmpLock	Yes/No			

Table 4.4: Hosts Table

Field Name	Data Type	Length	Index	F.K Base
CompName	Text	50	P.K.	
Host	Text	50		

Table 4.5: Prices Table

Field Name	Data Type	Length	Index	F.K Base
curname	Text	20	P.K.	
currprice	Number	Long Integer		

Table 4.6: Profile Table

Field Name	Data Type	Length	Index	F.K Base
account_no	Number	Long Integer	P.k.&F.k.	Users
email	Text	50		
SMTPHost	Text	50		
address	Text	50		
HTele	Text	20		
JTele	Text	20		
Company	Text	50		

Table 4.7: Services Table

Field Name	Data Type	Length	Index	F.K Base
invoiceno	Number	Long Integer	P.K.	
invoicetype	Text	10	P.K.	
account_no	Number	Long Integer		
amount	Number	Double		
payeddate	Text	30		

Table 4.8: Transfer Table

Field Name	Data Type	Length	Index	F.K Base
TID	Number	Long Integer	P.K.	
Facnt_no	Number	Long Integer		
Tacnt_no	Number	Long Integer		
amount	Number	Double		
pdate	Text	30		

Table 4.8: Users Table

Field Name	Data Type	Length	Index	F.K Base
<u>account_no</u>	Number	Long Integer	P.k.	
name	Text	50		
Password	Text	20		
amount	Number	Long Integer		
Lock	Yes/No			
created_date	Text	30		

Table 4.10: Withdraw Table

Field Name	Data Type	Length	Index	F.K Base T.
<u>WID</u>	Number	Long Integer	P.K.	
<u>account_no</u>	Number	Long Integer	P.K.	
Amount	Number	Double		
pdate	Text	30		



## CHAPTER FIVE

### INPUTS OUTPUTS DESIGN

#### 5.1 Sever Inputs Designs

##### 5.1.1 Server Main Form

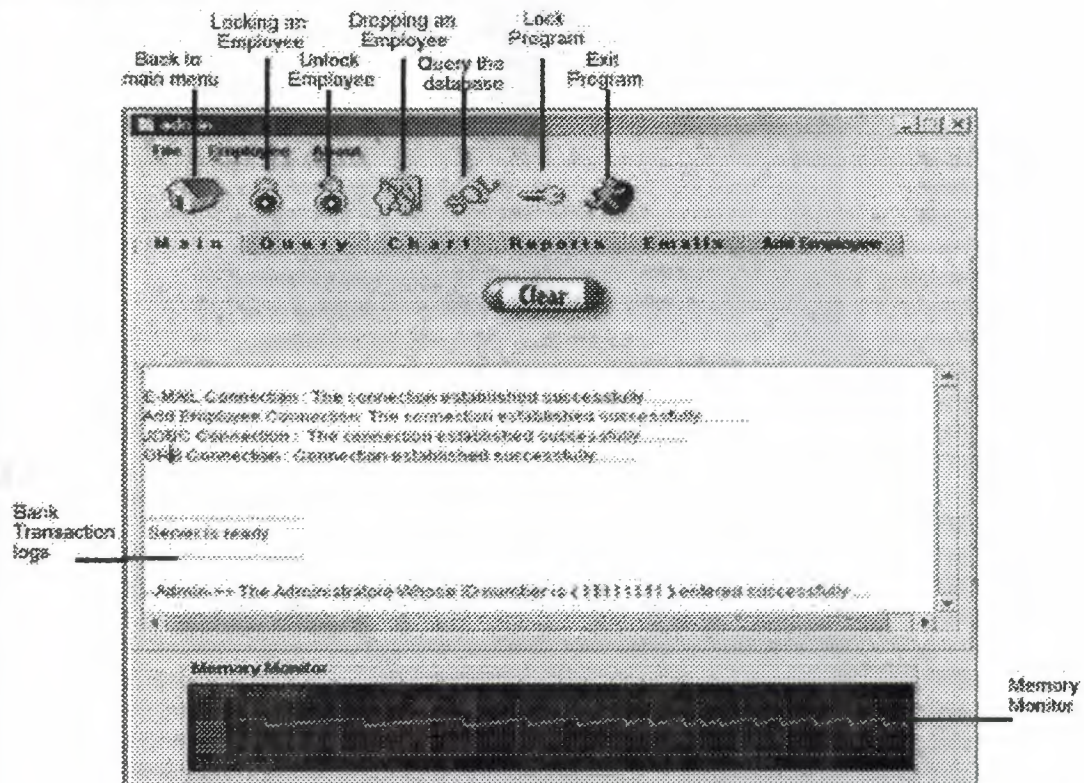


Figure 5.1: Server Main Form

Through the main server form the administrators can handle all administrative tasks such as bank transactions logging, database query, generating statistical bank charts, displaying status reports, sending emails and adding/dropping employees.

The File menu is used to lock or exit the program. And Employee menu is used to handle employees' tasks (locking/unlocking, adding/dropping). About menu gives brief information about the developer and program.

5.1.2 Server Query Form

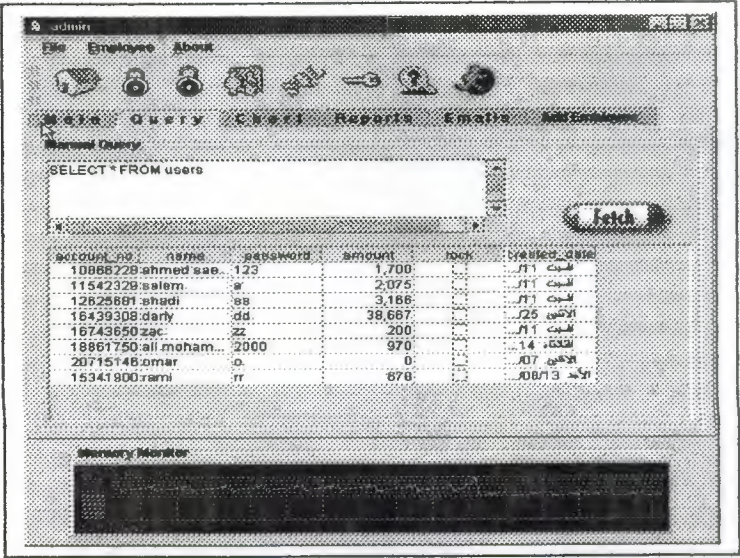


Figure 5.2: Server Query Form

5.1.3 Server Chart Form

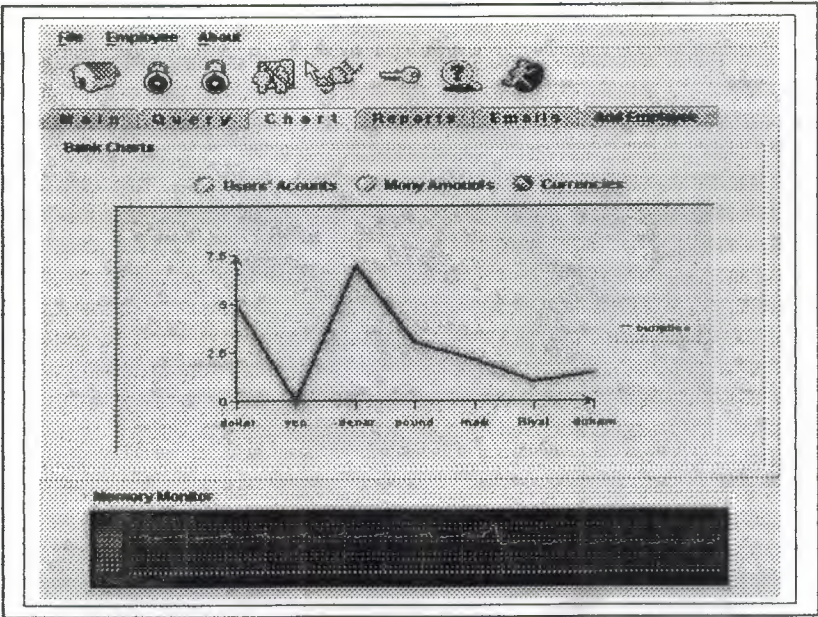


Figure 5.3: Server Chart Form



### 5.1.4 Server Reports Form

The screenshot shows a software window titled "File Employee About". Below the title bar is a toolbar with icons for file operations, a list of reports, a chart, and email functions. A menu bar contains "Main", "Query", "Chart", "Reports", "Emails", and "Add Employees". The "Reports" menu is open, displaying a list of "Bank Reports":

- ☒ Users' Details Report
- ☐ Employees Details Report
- ☐ Currencies
- ☐ Withdrawal Operations
- ☐ Deposit Operations
- ☐ Transfer Operations
- ☐ Invoice Operations

At the bottom of the report list is a "Preview" button. Below the menu is a "Memory Monitor" section with a graph showing memory usage over time.

Figure 5.4: Server Reports Form

### 5.1.5 Server E-Mails Form

The screenshot shows a software window titled "File Employee About". Below the title bar is a toolbar with icons for file operations, a list of reports, a chart, and email functions. A menu bar contains "Main", "Query", "Chart", "Reports", "Emails", and "Add Employees". The "Emails" menu is open, displaying a form for sending an email:

Select: 10000/220 - atendat.com - [v] ☐ All Users ☐ All Employees

To: [text box] @ yahoo.com [v]  
CC: [text box] @ yahoo.com [v]  
Subject: [text box]

[Send button]

Below the email form is a "Memory Monitor" section with a graph showing memory usage over time.

Figure 5.5: Server E-Mails Form

### 5.1.6 Server Add Employee Form

The screenshot shows a web-based application window titled "Server Add Employee Form". The interface includes a menu bar with "File", "Employee", and "About". Below the menu bar is a toolbar with icons for file operations, editing, and help. The main menu bar includes "Main", "Query", "Chart", "Reports", "Emails", and "Add Employee". The form fields include: Name (text box), Permissions (dropdown menu set to "Employee"), Password (text box) and Confirm (text box), E-mail (text box) and a domain dropdown set to "yahoo.com", Address (text box), Home Tel. (text box), and Job Tel. (text box). At the bottom are "Add", "Clear", and "Update" buttons. A "Memory Monitor" section is visible at the bottom of the window.

Figure 5.6: Server Add Employee Form



## 5.2 Employee Client Inputs Designs

### 5.2.1 Employee Client Main Form

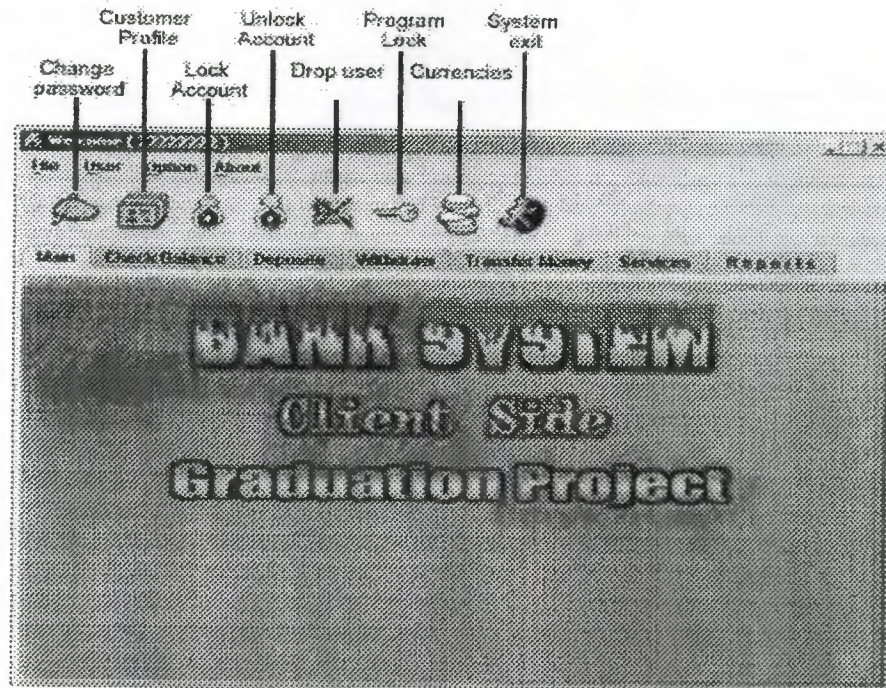


Figure 5.7: Employee Client Main Form

Through the main employee form the employees can handle all bank exchange tasks such as serving customers by checking their balances, deposit, withdraw transfer from or to their accounts, paying invoices and displaying account status reports. Employees can also add/drop, lock/unlock customer accounts and checking currencies exchange prices.

The File menu is used to lock or exit the program. And User menu is used to handle user' tasks (locking/unlocking, adding/dropping). Options Menu is used to change customers' passwords and profiles. About menu gives brief information about the developer and program.



### 5.2.2 Employee Client Check Balance Form

The screenshot shows a software window titled "Welcome (22222222)". It features a menu bar with "File", "User", "Option", and "About". Below the menu is a toolbar with icons for a speech bubble, calendar, padlock, key, crossed-out key, coffee cup, person, computer monitor, and a person with a speech bubble. A tabbed interface at the bottom includes "Main", "Check Balance", "Deposit", "Withdraw", "Transfer Money", "Services", and "Reports". The "Check Balance" tab is active, displaying a decorative header with a 3D effect and the text "Check Balance". The form contains a "Password" label followed by a text input field, a "Check" button, a "Status" label, and a large empty text area at the bottom.

Figure 5.8: Employee Client Check Balance Form

### 5.2.3 Employee Client Deposit Form

The screenshot shows a software window titled "Welcome (22222222)" with the same menu bar and toolbar as Figure 5.8. The "Deposit" tab is active in the tabbed interface. The form has a decorative header with the text "Deposit". It includes a "password" label with a text input field, an "Amount" label with a text input field, a "Deposit" button, an "ACCOUNT STATUS" label, and a large empty text area at the bottom.

Figure 5.9: Employee Client Deposit Form



### 5.2.4 Employee Client Withdraw Form

The screenshot shows a software window titled "Welcome (22222222)". It features a menu bar with "File", "User", "Option", and "About". Below the menu is a toolbar with icons for various functions. A navigation bar contains links: "Main", "Check Balance", "Deposits", "Withdraw", "Transfer Money", "Services", and "Reports". The "Withdraw" link is highlighted. The main content area has a decorative header with the word "Withdraw" on the right. Below this, there are two input fields labeled "password" and "Amount". A "Withdraw" button is positioned below these fields. At the bottom, there is a section labeled "Account Status" with a large empty rectangular box for displaying information.

Figure 5.10: Employee Client Withdraw Form

### 5.2.5 Employee Client Transfer Form

The screenshot shows a software window titled "Welcome (22222222)". It features a menu bar with "File", "User", "Option", and "About". Below the menu is a toolbar with icons for various functions. A navigation bar contains links: "Main", "Check Balance", "Deposits", "Withdraw", "Transfer Money", "Services", and "Reports". The "Transfer Money" link is highlighted. The main content area has a decorative header with the words "Money Transfer" on the right. Below this, there are three input fields: "From Account: 22222222" (pre-filled), "Password", and "To Account". A fourth input field labeled "Amount" is below the "To Account" field. A "Transfer" button is positioned below these fields. At the bottom, there is a section labeled "Account Status" with a large empty rectangular box for displaying information.

Figure 5.11: Employee Client Transfer Form



### 5.2.6 Employee Client Services Form

The screenshot shows a software window titled "Welcome (22222222)". It features a menu bar with "File", "User", "Option", and "About". Below the menu bar is a toolbar with icons for various functions. A tabbed interface at the top includes "Main", "Check Balance", "Deposits", "Withdraw", "Transfer Money", "Services", and "Reports". The "Services" tab is currently selected, displaying a section titled "Services". Under this section, there are three categories: "Electricity", "Water", and "Telephon". Each category has two input fields: "Invoice No" and "Invoice Value". To the right of each category's input fields is a "Pay" button. The "Electricity" section has a "Pay" button, the "Water" section has a "Pay" button, and the "Telephon" section has a "Pay" button.

Figure 5.12: Employee Client Services Form

### 5.2.7 Employee Client Reports Form

The screenshot shows a software window titled "Welcome (22222222)". It features a menu bar with "File", "User", "Option", and "About". Below the menu bar is a toolbar with icons for various functions. A tabbed interface at the top includes "Main", "Check Balance", "Deposits", "Withdraw", "Transfer Money", "Services", and "Reports". The "Reports" tab is currently selected, displaying a section titled "Reports". Under this section, there is a "Bank Reports" section with five radio button options: "Withdraw Operations", "Deposit Operations", "Outcome Transfer Operations", "Income Transfer Operations", and "Invoice Operations". The "Withdraw Operations" option is selected. At the bottom left of the "Reports" section is a "Preview" button.

Figure 5.13: Employee Client Reports Form



### 5.2.8 Employee Client New User Form

The screenshot shows a web application window titled "NewUser Menu". Inside, there is a "New User" form. The form includes the following fields: "user name", "password", "Confirm", "E-Mail" (with a dropdown menu showing "yahoo.com"), "Address", "Home Tel.", "Job Tel.", and "Company". Below these fields is a "Sign-In" button. At the bottom of the form, there is a "Reply Message:" label and a large text area. To the right of the text area, there is a "connection" section with the text "to connect with your new account click here ..." and a "Connect" button.

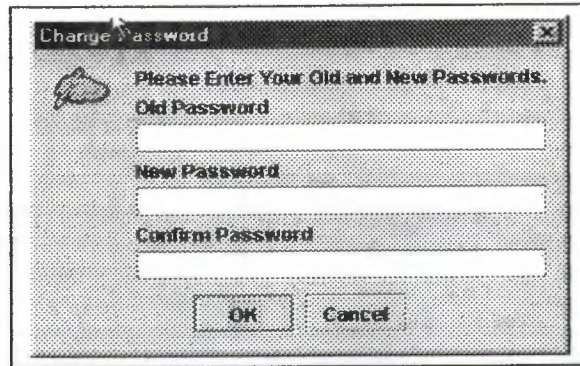
Figure 5.14: Employee Client New User Form

### 5.2.9 Employee Client Update User Profile Form

The screenshot shows a web application window titled "Profile". Inside, there is a "User Profile" form. The form includes the following fields: "Account NO." (with value "15311800"), "user name" (with value "raml"), "E-Mail" (with value "ramico2001" and a dropdown menu showing "yahoo.com"), "SMTP Host" (with value "mx1.mail.yahoo.com"), "Address" (with value "gaza"), "Home Tel." (with value "2807656"), "Job Tel." (with value "2806576"), and "Company" (with value "azhar"). Below these fields, there is a message: "Press Update profile to update changes or OK to exit." and two buttons: "OK" and "Update Profile".

Figure 5.15: Employee Client Update User Profile Form

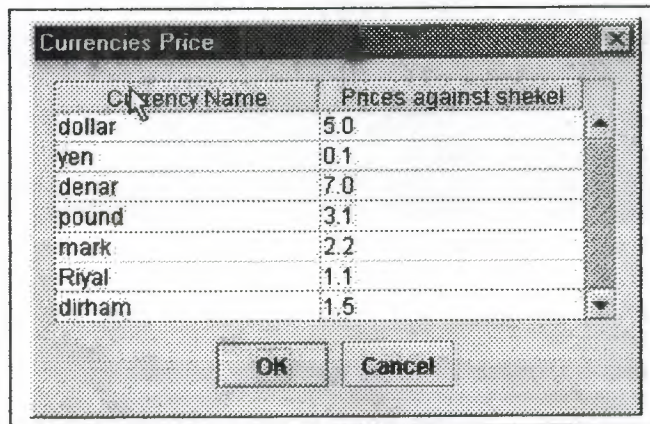
### 5.2.10 Employee Client Change User Password Form



A dialog box titled "Change Password" with a close button (X) in the top right corner. It contains a small icon of a person on the left and the text "Please Enter Your Old and New Passwords." followed by three input fields labeled "Old Password", "New Password", and "Confirm Password". At the bottom are "OK" and "Cancel" buttons.

Figure 5.16: Employee Client Change User Password Form

### 5.2.11 Employee Client Show Currencies Form



A dialog box titled "Currencies Price" with a close button (X) in the top right corner. It contains a table with two columns: "Currency Name" and "Prices against shekel". The table lists several currencies and their corresponding prices. At the bottom are "OK" and "Cancel" buttons.

Currency Name	Prices against shekel
dollar	5.0
yen	0.1
denar	7.0
pound	3.1
mark	2.2
Riyal	1.1
dirham	1.5

Figure 5.17: Employee Client Show Currencies Form



## 5.3 User Client Inputs Designs

### 5.3.1 User Client Main Form

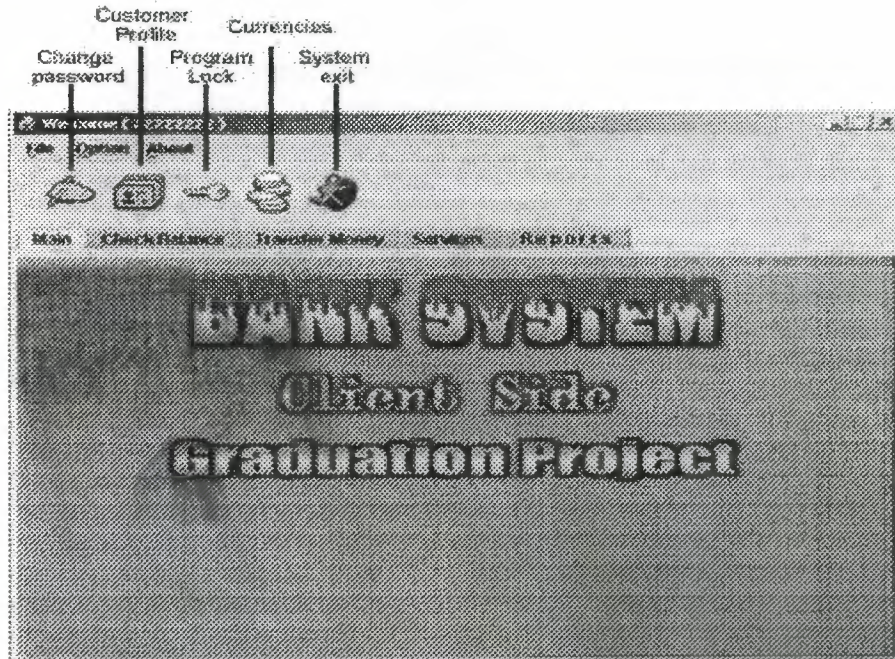


Figure 5.18: User Client Main Form

Through the main Customers' form the customers can access bank resources and perform specific tasks such as checking their balances, transferring from their accounts, paying invoices and displaying account status reports. Customers can also check currencies exchange prices.

The File menu is used to lock or exit the program. Options Menu is used to change customers' passwords and profiles. About menu gives brief information about the developer and program.



3.3.2 User Client Check Balance Form

Welcome (22222222)

File Option About

Main Check Balance Transfer Money Services Reports

Check Balance

Password

Check

Status

Figure 5.19: User Client Check Balance Form

5.3.3 User Client Transfer Money Form

Welcome (22222222)

File Option About

Main Check Balance Transfer Money Services Reports

Money Transfer

From Account 22222222

Password

To Account

Amount

Transfer

Account Status

Figure 5.20: User Client Transfer Money Form



### 5.3.4 User Client Services Form

The screenshot shows a web application window titled "Welcome (22222222)". The menu bar includes "File", "Option", and "About". Below the menu bar are several icons: a speech bubble, a calendar, a key, a coffee cup, a person, and a star. The main navigation tabs are "Main", "Check Balance", "Transfer Money", "Services", and "Reports". The "Services" tab is currently selected, and the page title "Services" is displayed on the right. The form is divided into three sections: "Electricity", "Water", and "Telephon". Each section contains two input fields: "Service No" and "Service Value". To the right of each section is a "Pay" button.

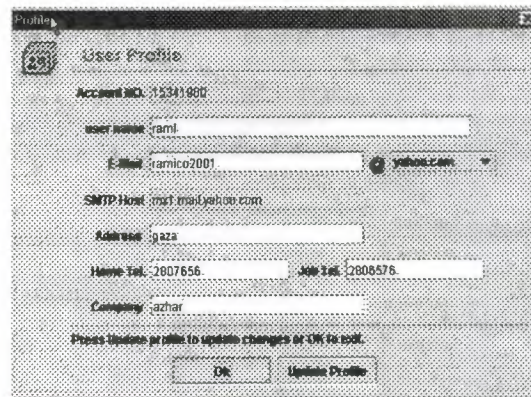
Figure 5.21: User Client Services Form

### 5.3.5 User Client Reports Form

The screenshot shows a web application window titled "Welcome (22222222)". The menu bar includes "File", "Option", and "About". Below the menu bar are several icons: a speech bubble, a calendar, a key, a coffee cup, a person, and a star. The main navigation tabs are "Main", "Check Balance", "Transfer Money", "Services", and "Reports". The "Reports" tab is currently selected, and the page title "Reports" is displayed on the right. The form is titled "Bank Reports" and contains five radio button options: "Withdraw Operations", "Deposit Operations", "Outgoing Transfer Operations", "Income Transfer Operations", and "Invoice Operations". A "Preview" button is located at the bottom left of the form.

Figure 5.22: User Client Reports Form

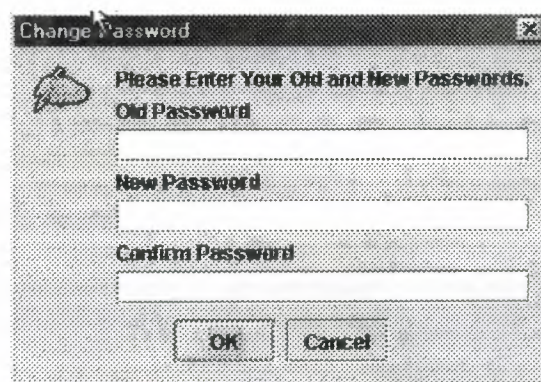
### 5.3.6 User Client Update User Profile Form



A screenshot of a 'User Profile' dialog box. It contains several input fields: 'Account ID' (12341800), 'User Name' (rami), 'E-Mail' (ramico2001), 'SMTP Host' (mail.mai.yahoo.com), 'Address' (paza), 'Home Tel.' (2807656), 'Mob Tel.' (2808576), and 'Company' (azhar). At the bottom, there is a message 'Press Update profile to update changes or OK to exit.' and two buttons: 'OK' and 'Update Profile'.

Figure 5.23: User Client Update User Profile Form

### 5.3.7 User Client Change User Password Form



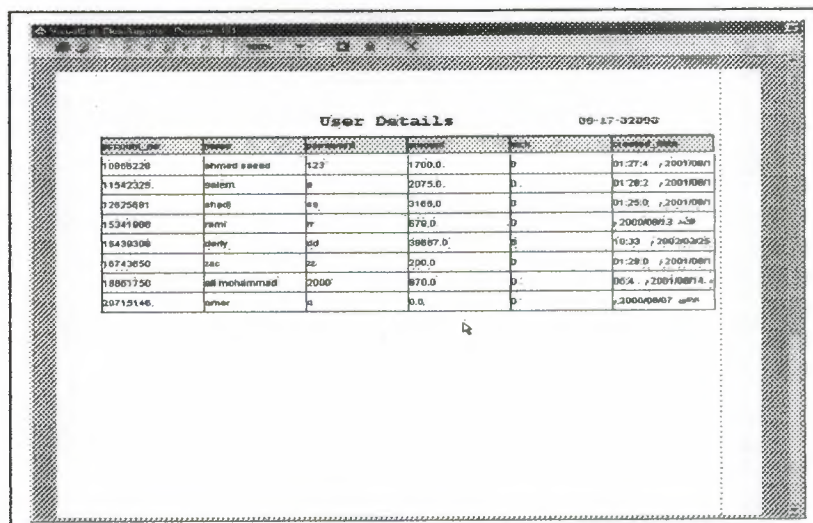
A screenshot of a 'Change Password' dialog box. It contains three input fields: 'Old Password', 'New Password', and 'Confirm Password'. At the bottom, there are two buttons: 'OK' and 'Cancel'.

Figure 5.24: User Client Change User Password Form



## 5.4 Server Outputs Designs:

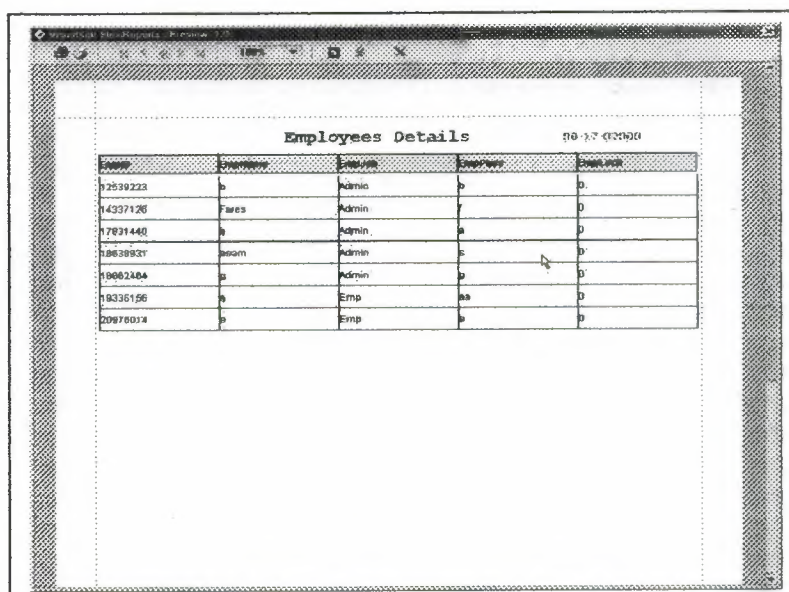
### 5.4.1 Users Details Report



Username	Password	Username	Password	Salt	Created Date
10802220	ahmed saad	123	1700.0	0	01:27:4 , 2001/08/1
11542325	salim	0	2075.0	0	01:28:2 , 2001/08/1
12625881	ahed	00	2165.0	0	01:29:0 , 2001/08/1
15341000	amr	0	870.0	0	, 2000/08/13 4:39
15430308	sami	00	39887.0	0	10:33 , 2002/03/25
16743050	samy	00	200.0	0	01:28:0 , 2001/08/1
18801750	sac	00	870.0	0	05:4 , 2001/08/1 8:4
20715140	ali mohammad	2000	0.0	0	, 2000/08/07 00:00

Figure 5.25: Users Details Report

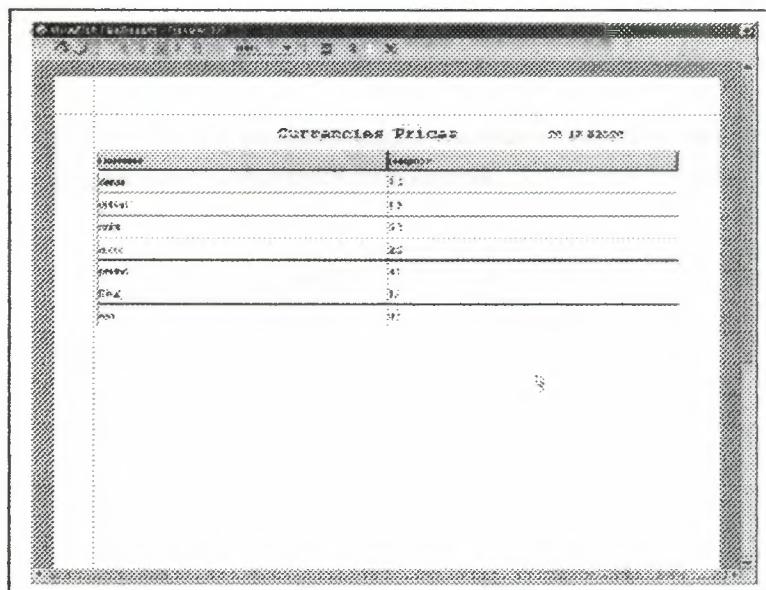
### 5.4.2 Employees Details Report



Employee	Employee	Employee	Employee	Employee
12539223	b	Admin	b	0
14237120	Fares	Admin	0	0
17021440	Admin	Admin	0	0
18638937	Admin	Admin	0	0
19002404	Admin	Admin	0	0
19336156	Emp	Emp	0	0
20970014	Emp	Emp	0	0

Figure 5.26: Employees Details Report

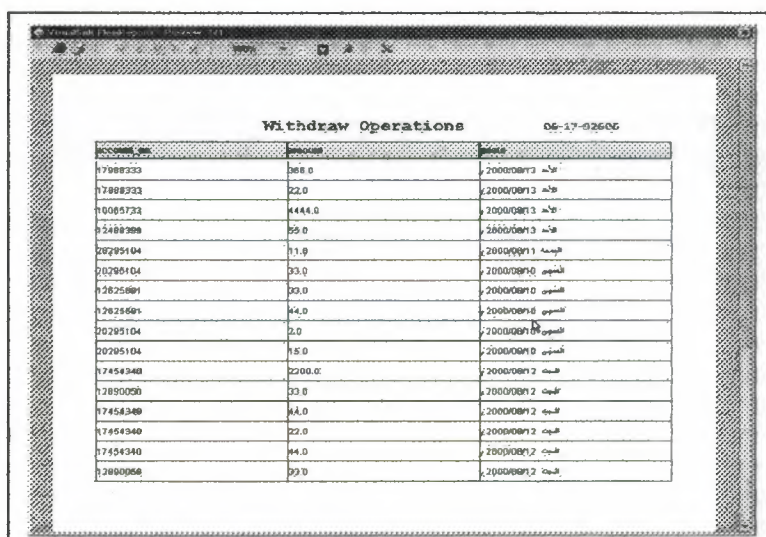
### 5.4.3 Currencies Prices Report



Currency	Price
USD	1.0
EUR	0.75
GBP	0.65
JPY	110.0
AUD	0.75
NZD	0.65
CAD	0.75

Figure 5.27: Currencies Prices Report

### 5.4.4 Withdraw Operations Report



Account	Amount	Currency
1700000000	100.0	USD
1700000000	22.0	USD
1000000000	4444.0	USD
1700000000	55.0	USD
20295104	11.0	USD
20295104	33.0	USD
1700000000	23.0	USD
1700000000	44.0	USD
20295104	2.0	USD
20295104	15.0	USD
17454340	2200.0	USD
1700000000	23.0	USD
17454340	44.0	USD
17454340	22.0	USD
17454340	44.0	USD
1700000000	23.0	USD

Figure 5.28: Withdraw Operations Report



### 5.4.5 Deposit Operations Report

Account No	Amount	Date
12406399	55.0	2000/08/3
17988333	44.0	2000/08/12
10005233	4444.0	2000/08/3
15341900	777.0	2000/08/6
20295104	33.0	2000/08/11
16438308	38887.0	2000/08/11
20295104	33.0	2000/08/10
20295104	55.0	2000/08/10
17454340	44.0	2000/08/12
20295104	55.0	2000/08/12
20295104	55.0	2000/08/12
17880050	33.0	2000/08/12
17454340	200.0	2000/08/12
17454340	22.0	2000/08/12
17454340	44.0	2000/08/12
20295104	44.0	2000/08/12
17980050	33.0	2000/08/12

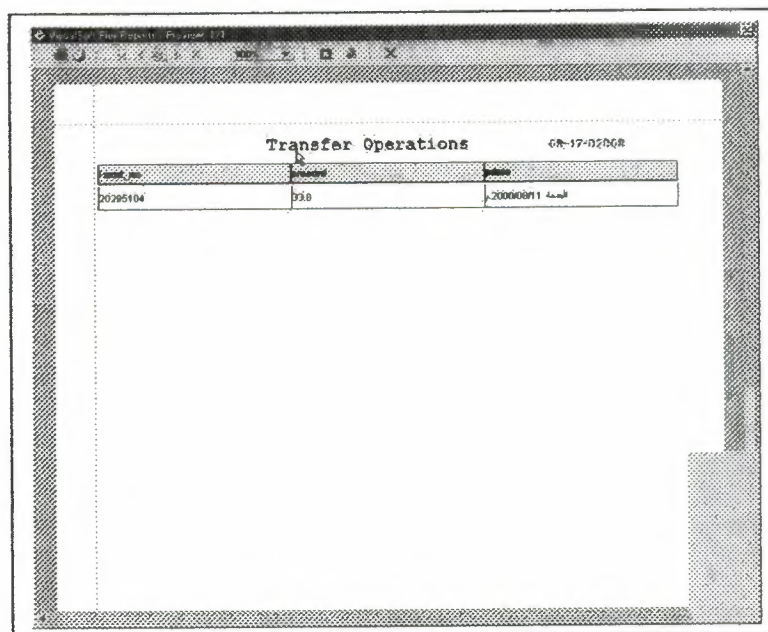
Figure 5.29: Deposit Operations Report

### 5.4.6 Export Transfer Operations Report

Account No	Amount	Date
46743650	33.0	2000/08/11

Figure 5.30: Export Transfer Operations Report

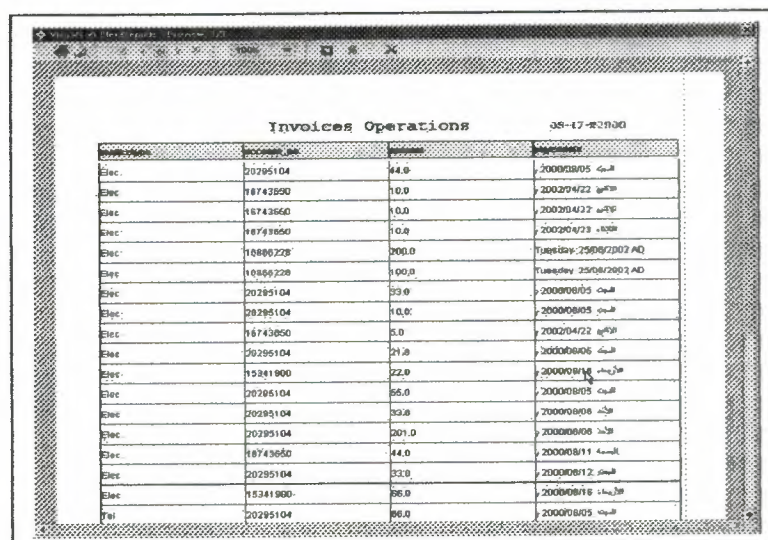
### 5.4.7 Import Transfer Operations Report



Transfer No	Transfer Amount	Transfer Date
20295104	33.8	20080811

Figure 5.31: Import Transfer Operations Report

### 5.4.8 Invoices Operations Report



Invoice No	Invoice Amount	Invoice Date	Invoice Description
Elec.	20295104	44.8	20080805
Elec.	18743850	10.0	20020422
Elec.	18743850	10.0	20020422
Elec.	18743850	10.0	20020422
Elec.	18743850	10.0	20020422
Elec.	18886228	200.0	Tuesday 25/08/2002 AD
Elec.	18886228	100.0	Tuesday 25/08/2002 AD
Elec.	20295104	33.8	20080805
Elec.	20295104	10.0	20080805
Elec.	18743850	5.0	20020422
Elec.	20295104	21.8	20080805
Elec.	15281980	22.8	20080811
Elec.	20295104	65.0	20080805
Elec.	20295104	33.8	20080805
Elec.	20295104	201.0	20080805
Elec.	18743850	44.0	20080811
Elec.	20295104	33.8	20080812
Elec.	15281980	86.0	20080816
Elec.	20295104	86.0	20080805

Figure 5.32: Invoices Operations Report

5.5 Clients Outputs Designs:

5.5.1 User Withdraw Operations Report

Amount	Balance
77.0	280000017
432.0	280000017
800.0	280000017
22.0	280000017
44.0	280000017

Figure 5.33: User Withdraw Operations Report

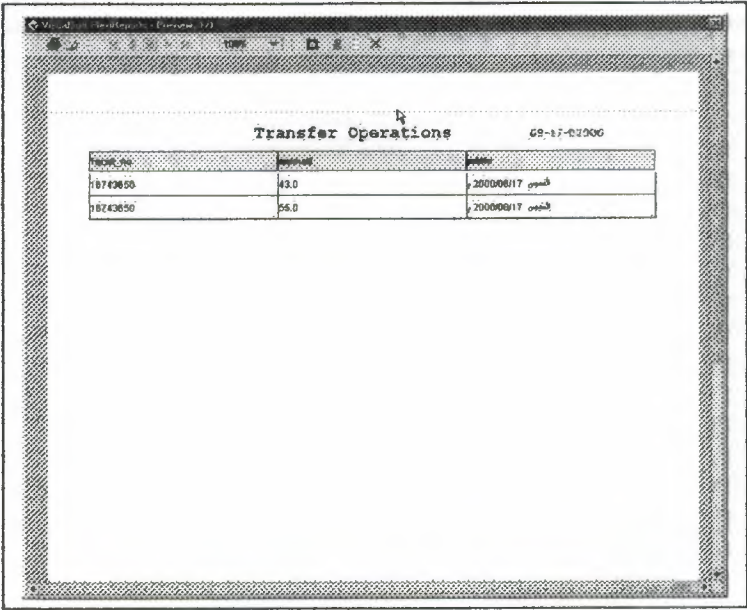
5.5.2 User Deposit Operations Report

Amount	Balance
777.0	280000017
1111.0	280000017
800.0	280000017
87.0	280000017
199.0	280000017
44.0	280000017

Figure 5.34: User Deposit Operations Report



5.5.3 User Export Transfer Operations Report

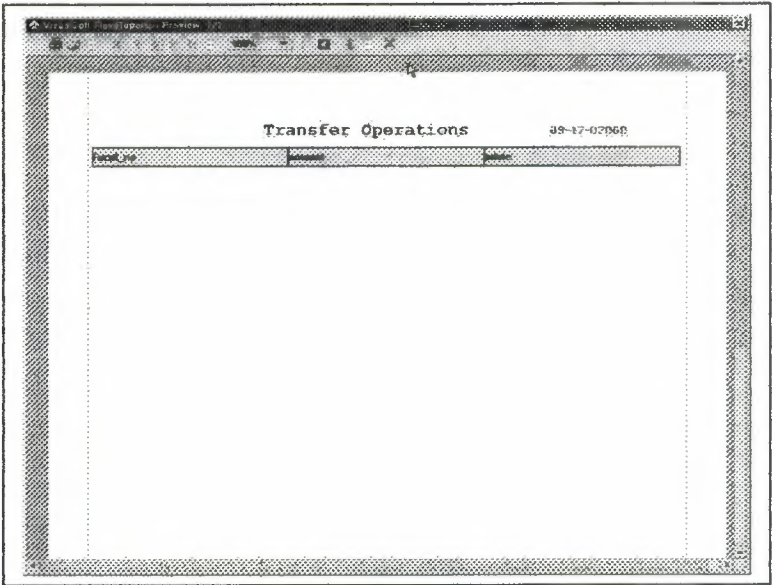


The screenshot shows a Microsoft Word document window titled 'Transfer Operations'. The document contains a table with three columns: 'Transfer No.', 'Amount', and 'Date'. The first row shows '18742650', '43.0', and '200008/17'. The second row shows '18742650', '56.0', and '200009/17'. The document is dated '09-17-02'.

Transfer No.	Amount	Date
18742650	43.0	200008/17
18742650	56.0	200009/17

Figure 5.35: User Export Transfer Operations Report

5.5.4 User Import Transfer Operations Report



The screenshot shows a Microsoft Word document window titled 'Transfer Operations'. The document contains a table with three columns: 'Transfer No.', 'Amount', and 'Date'. The table is currently empty. The document is dated '09-17-02'.

Transfer No.	Amount	Date
--------------	--------	------

Figure 5.36: User Import Transfer Operations Report



5.5.5 User Invoices Operations Report

Invoice No.	Invoice Date	Invoice Amount
20000877	2000/08/16	22.0
20000876	2000/08/16	22.0
20000877	2000/08/16	22.0
20000877	2000/08/16	22.0
20000877	2000/08/16	22.0
20000876	2000/08/16	22.0
20000876	2000/08/16	22.0

Figure 5.37: User Invoices Operations Report

## **CONCLUSION**

Through out this project I've introduced a solution for handling most common bank systems' tasks by developing a practicable easy-to-use Graphical User Interface (GUI).

Distributing the bank system makes it more reliable, expandable and flexible. So, it could handle many client transactions as well as server transactions at the same time with a high performance; since it is also distribute the load among all communication terminals.

This system could also be used as centralized administration point for all bank clients since it introduces a complete logging and tracing for all tasks done by both employees and customers. And also it introduces the most common administrative tasks through a graphical interface to simplify the administrators' works.

## REFERENCES

- [1] Jeremy Rosenberger, Teach Yourself CORBA in 14 Days. Macmillan Computer Publishing.
- [2] Mark Grand and Jonathan Knudsen, Java Fundamental Classes Reference.
- [3] <http://java.sun.com>
- [4] <http://www.informit.com>
- [5] Java 2 Standard Edition SDK 1.4.0 documentations
- [6] <http://www.JavaWorld.com>



## APPENDIX A

### PROGRAM SOURCE CODE

#### Client Class

```

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
import javax.swing.border.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.*;

import java.util.*;
import java.io.*;
import java.net.*;

// corba packages
import BankApp.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
//-----//
//
//                               The Frame Paint functions
//
//-----//
//CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
//    The Client class
//CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
public class Client {

    private static JWindow splashScreen = null;
    private static JLabel splashLabel = null;
    private static javax.swing.Timer t,tt;
    static JFrame fr;

    static JProgressBar progressBar;
    static JLabel progressLabel;

    //-----//
    //    Function name:  the StartSplashScreen function
    //    Function Work:  show the Splash screen in the first loading of the program
    //                   then hide it after 8 second then show the program frame
    //-----//
    public static void StartSplashScreen(){
        // the dealay time to present the splash screen
        int ONE_SECOND = 3200; // the presentation of the splash delay
        //create timer to wait a specific time
        t = new javax.swing.Timer(ONE_SECOND, new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                hideSplash(); //to hide the splash screen after the delay
            }
        });
    }

```

```

        PLogin();
    });
    createSplashScreen(); // creat the splash screen
    showSplashScreen(); // show the Splash Screen
    t.start();           //start the timer
    return ;
} // end of function

//-----
//      Function name:  the createSplashScreen function
//      Function Work:  create the splash screen by using JWindow function
//                      then put it position to the center of the screen
//-----
public static void createSplashScreen() {

    splashLabel = new JLabel(new ImageIcon("image/Splash.jpg"));

    splashScreen = new JWindow();
    splashScreen.getContentPane().add(splashLabel);
    splashScreen.pack();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    splashScreen.setLocation(screenSize.width/2 - splashScreen.getSize().width/2,
                           screenSize.height/2 - splashScreen.getSize().height/2);
} // end of function

//-----
//      Function name:  the showSplashScreen function
//      Function Work:  pop up the splash screen
//-----
public static void showSplashScreen() {
    splashScreen.show();
} // end of function

//-----
//      Function name:  the hideSplash function
//      Function Work:  pop down the splash screen
//-----
public static void hideSplash() {
    splashScreen.setVisible(false);
    splashScreen = null;
    splashLabel = null;
    t.stop();
} // end of hideSplash function

//-----
//      Function name:  the PLogin function
//      Function Work:  Pefore Login frame
//-----
public static void PLogin(){
    JFrame f = new JFrame("null");
    JPanel chkBox = new JPanel();
    chkBox.setLayout(new BoxLayout(chkBox, BoxLayout.X_AXIS));
    JCheckBox user= new JCheckBox(" User ");
    JCheckBox emp= new JCheckBox(" Employee ");

```

```

JLabel orlbl =new JLabel(" OR ");
JLabel gab =new JLabel(" ");
chkBox.add(gab);
chkBox.add(user);
chkBox.add(orbitl);
chkBox.add(emp);

String message = " Please ! Check your Classification , Then click OK ... ";
Imagelcon loginlcon = new Imagelcon("image/login.gif");

int result = JOptionPane.showOptionDialog(f,new java.lang.Object[]{ message,chkBox},
                                         "Login",JOptionPane.OK_CANCEL_OPTION
                                         ,
                                         JOptionPane.QUESTION_MESSAGE,
                                         loginlcon ,new
                                         java.lang.Object[]{"OK","Cancel"}, null);

int x =110;
if (result == 0)
{
    if (user.isSelected() && emp.isSelected())
    {
        JOptionPane.showMessageDialog(f,"Please! Select Login as User or Employee
only");
        PLogin();
    }
    else if (!user.isSelected() && !emp.isSelected())
    {
        JOptionPane.showMessageDialog(f,"Please! Select one of ( User and Employee
)");
        PLogin();
    }
    else if (user.isSelected())
    {
        UserFrame uf;
        uf = new UserFrame();
    }
    else
    {
        EmpFrame ef;
        ef = new EmpFrame();
    }
}
}else if (result == 1){
    System.exit(0);}

}

//-----
//      Function name:  the progres function
//      Function Work:  give more safty to the client user by locking
//                      the program when he leave it
//-----
public static void progres(){

fr = new JFrame("Distributed Banking System");

```



```

fr.getAccessibleContext().setAccessibleDescription("A sample application to demonstrate
Java2D features");
int WIDTH = 400, HEIGHT = 200;
fr.setSize(WIDTH, HEIGHT);
Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
fr.setLocation(d.width/2 - WIDTH/2, d.height/2 - HEIGHT/2);
fr.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
fr.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {System.exit(0);}
});
JOptionPane.setRootFrame(fr);
    JPanel progressPanel = new JPanel() {
        public Insets getInsets() {
            return new Insets(40,30,20,30);
        }
    };
progressPanel.setLayout(new BoxLayout(progressPanel, BoxLayout.Y_AXIS));
fr.getContentPane().add(progressPanel, BorderLayout.CENTER);

Dimension labelSize = new Dimension(400, 20);
progressLabel = new JLabel("Loading, please wait...");
progressLabel.setAlignmentX(SwingConstants.CENTER);
progressLabel.setMaximumSize(labelSize);
progressLabel.setPreferredSize(labelSize);
progressPanel.add(progressLabel);
progressPanel.add(Box.createRigidArea(new Dimension(1,20)));

progressBar = new JProgressBar();
progressBar.setStringPainted(true);
progressLabel.setLabelFor(progressBar);
progressBar.setAlignmentX(SwingConstants.CENTER);
progressBar.setMinimum(0);
progressBar.setValue(0);
progressBar.getAccessibleContext().setAccessibleName("Java2D loading progress");
progressPanel.add(progressBar);

fr.setVisible(true);
//fr.getContentPane().removeAll();
fr.getContentPane().setLayout(new BorderLayout());
fr.validate();
fr.repaint();
}

//~~~~~
// main *****
//~~~~~
public static void main(String[] args) {

    progres();
    progressBar.setMaximum(13);
    progressLabel.setText("Loading images");
    progressBar.setValue(progressBar.getValue() + 1);
    progressLabel.setText("Loading menus");
    progressBar.setValue(progressBar.getValue() + 1);

```

```
progressLabel.setText("Loading Please wait ....");
progressBar.setValue(progressBar.getValue() + 1);
progressBar.setValue(progressBar.getValue() + 1);
progressBar.setValue(progressBar.getValue() + 1);
progressBar.setValue(progressBar.getValue() + 1);
progressBar.setValue(progressBar.getValue() + 1);
progressBar.setValue(progressBar.getValue() + 1);
progressBar.setValue(progressBar.getValue() + 1);
progressBar.setValue(progressBar.getValue() + 10);
```

```
int ONE_SECOND = 1000; // the presentation of the splash delay
//create timer to wait a specific time
tt = new javax.swing.Timer(ONE_SECOND, new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        fr.setVisible(false);
    }
});

tt.start(); //start the timer

StartSplashScreen();
}

}
```

[illegible]































```

String res1 = "Account is not found" ;
double newamount1;
int z=4;
try{

    String QU1 ="Select account_no,amount From users where account_no
="+AC1;
    ResultSet RS1 = stmt.executeQuery(QU1);
    while(RS1.next()){
        int Acnt1=RS1.getInt("account_no");
        double Ant1=RS1.getDouble("amount");
        int acoul = new Integer(AC1).intValue();
        double amoul = new Double(AM1).doubleValue();
        int inn = new Integer(INV).intValue();
        String ty1="Elec";
        String ty2="Wtr";
        String ty3="Tel";
        if(Acnt1==acoul){
            String testQ ="Select invoiceno,invoicetype
From services where account_no ="+"AC1;
            ResultSet testRS = stmt.executeQuery(testQ);
            while(testRS.next()){
                String NO =
testRS.getString("invoiceno");
                String TYPE =
testRS.getString("invoicetype");

                if ( (NO.equals(INV)) &&
(TYPE.equals(type)) )    //(NO == inn)
                {
                    res1="Fail operation.. \nThis
Invoice Already Payed ";
                    z = 122;
                }
            }//end of 2nd while

            if (z==122)
            {
            }
            else if(Ant1==0){
                res1="Fail operation.. \nyour account is
empty ";
                BankServer.printMsg("\n[ Pay Invoice ]> !!
Error : Fail pay the "+type+" Invoice operation.....for the User [
"+AC1+" ] has an empty account ....");
            }
            else if(Ant1<amoul){
                res1="Fail operation.. \nthe balance is not
enough ";
                BankServer.printMsg("\n[ Pay Invoice ]> !!
Error : Fail pay the "+type+" Invoice operation.....for the User [
"+AC1+" ] has not enough account ....");
            }
            else{
                try{

```



```

        newamount1 = Ant1-amoul;
        String Qul="Update users set
amount = "+String.valueOf(newamount1)+" where account_no ="+AC1;
        res1=" your new balance is
"+String.valueOf(newamount1)+" $";
        if(ty1.equals(type)){
            String q1="insert into
services(account_no,invoiceno,invoicetype,amount,payeddate)
values("+AC1+", "+INV+", '"+ty1+"', "+amoul+", '"+getDate()+"'");";
            stmt.executeUpdate(q1);
            BankServer.printMsg("\n[
Pay Invoice ]> The Customer [ "+AC1+" ] is pay a Electric Invoice
successfully ....");
        }
        else if(ty2.equals(type)){
            String q2="insert into
services(account_no,invoiceno,invoicetype,amount,payeddate)
values("+AC1+", "+inn+", '"+ty2+"', "+amoul+", '"+getDate()+"'");";
            stmt.executeUpdate(q2);
            BankServer.printMsg("\n[
Pay Invoice ]> The Customer [ "+AC1+" ] is pay a Water Invoice
successfully ....");
        }
        else if(ty3.equals(type)){
            String q3="insert into
services(account_no,invoiceno,invoicetype,amount,payeddate)
values("+AC1+", "+inn+", '"+ty3+"', "+amoul+", '"+getDate()+"'");";
            stmt.executeUpdate(q3);
            BankServer.printMsg("\n[
Pay Invoice ]> The Customer [ "+AC1+" ] is pay a Telephone Invoice
successfully ....");
        }
    }
    stmt.executeQuery(Qul);
} //end of try
catch (SQLException ex){
    System.out.println ("SQLException:
Error In Invoice Function >>..>>..>>..>>");
    while (ex != null){
        System.out.println ("SQLState:
" + ex.getSQLState());
        System.out.println
("Message: " + ex.getMessage());
        System.out.println
("Vendor: " + ex.getErrorCode());
        ex =
ex.getNextException();
        System.out.println ("");
    }
} //end od catch
} //end of else
} //end of if
} //end of while
} //end of rty
catch(SQLException sw2){};
return res1;
} //end of inviiice

```



















```
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>
//      Function name: the DepositOp function
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>
public void DepositOp(String acct)
{
    Vector fieldsVector2 = new Vector();
    Vector sortFieldsVector2=new Vector();
    Vector sortFieldsType2 = new Vector();

    fieldsVector2.addElement("deposit1.amount");
    fieldsVector2.addElement("deposit1.pdate");
    sortFieldsVector2.addElement("deposit1.pdate");
    sortFieldsType2.addElement("Asc");
    String FilterStr="deposit1.account_no = "+acct;
    String RptHdr= " Deposit Operations ";

        CreateReport r = new CreateReport(fieldsVector2,
sortFieldsVector2, sortFieldsType2, FilterStr, RptHdr);
}
```

```
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>
//      Function name: the ETransferOp function
//>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>
public void ETransferOp(String acct)
{
    Vector fieldsVector2 = new Vector();
    Vector sortFieldsVector2=new Vector();
    Vector sortFieldsType2 = new Vector();

    fieldsVector2.addElement("transferl.Tacct_no");
    fieldsVector2.addElement("transferl.amount");
    fieldsVector2.addElement("transferl.pdate");
    sortFieldsVector2.addElement("transferl.pdate");
    sortFieldsType2.addElement("Asc");
    String FilterStr="transferl.Facct_no = "+acct;
    String RptHdr= " Transfer Operations ";

        CreateReport r = new CreateReport(fieldsVector2,
sortFieldsVector2, sortFieldsType2, FilterStr, RptHdr);
}
```

89





















```

        EServicesMenu.setToolTipText("Employee Services");

        JMenu m2 = new JMenu("About");
        m2.setMnemonic('a');
        m2.setToolTipText("About");

        JMenuItem i1 = new JMenuItem("Lock Server");
        i1.setMnemonic('l');
        i1.setIcon(new ImageIcon("image/ToolBar/lock.gif"));
        i1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
{lockServer();}}});

        JMenuItem i2 = new JMenuItem("Quit");
        i2.setMnemonic('q');
        i2.setIcon(new ImageIcon("image/ToolBar/exit.gif"));
        i2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
{validateExit();}}});

        JMenuItem lockID = new JMenuItem("Lock Employee");
        lockID.setMnemonic('k');
        lockID.setIcon(new ImageIcon("image/lock.gif"));
        lockID.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
{lockID_Dialog{0}; }}});

        JMenuItem unlockID = new JMenuItem("Unlock Employee");
        unlockID.setMnemonic('u');
        unlockID.setIcon(new ImageIcon("image/unlock.gif"));
        unlockID.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
{lockID_Dialog{1}; }}});

        JMenuItem dropEmp = new JMenuItem("Drop Employee");
        dropEmp.setMnemonic('e');
        dropEmp.setIcon(new ImageIcon("image/ToolBar/user1.gif"));
        dropEmp.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {dropEmpDialog();
}}});

        JMenuItem i3 = new JMenuItem("Bank Info.");
        i3.setMnemonic('b');
        i3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ImageIcon logoImage = new
ImageIcon("image/Splash_2.jpg");
                int result1 =
JOptionPane.showOptionDialog(null,new java.lang.Object[] {},
                                "Bank Information",
JOptionPane.CLOSED_OPTION ,
                                JOptionPane.QUESTION_MESSAGE,
                                logoImage, null, null);
            }
        });

        JMenuItem i4 = new JMenuItem("Designers");
        i4.setMnemonic('d');

```

```

        i4.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ImageIcon logoImage = new
ImageIcon("image/design1.jpg");
                int result1 =
OptionPane.showOptionDialog(null,new java.lang.Object[] {},
                                "Designers",
OptionPane.CLOSED_OPTION ,
                                JOptionPane.QUESTION_MESSAGE,
                                logoImage, null, null);
            }
        });

        m1.add(i1);
        m1.add(i2);
        EServicesMenu.add(lockID);
        EServicesMenu.add(unlockID);
        EServicesMenu.add(dropEmp);
        m2.add(i3);
        m2.add(i4);
        bar.add(m1);
            bar.add(EServicesMenu);
            bar.add(m2);
            serverf.setJMenuBar(bar);

JPanel CPanel = new JPanel();
CPanel.add(ChartPanel());

JPanel QueryPanel = new JPanel();
QueryPanel.add(UserPanel());

JPanel MainPanel = new JPanel();
MainPanel.add(GetMainPanel());

JPanel RPanel = new JPanel();
RPanel.add(ReportPanel());

JPanel EPanel = new JPanel();
EPanel.add(EmailPanel());

JPanel AddEPanel = new JPanel();
AddEPanel.add(AddEmpPanel());

tabby.addTab(" M a i n ",null,MainPanel);
tabby.addTab(" Q u e r y ",null,QueryPanel,"Write any SQL and fetch
the table");
tabby.addTab(" C h a r t ",null,CPanel);
tabby.addTab(" R e p o r t s ",null,RPanel);
tabby.addTab(" E m a i l s ",null,EPanel);
tabby.addTab(" Add Employee ",null,AddEPanel);

tabby.setSelectedComponent(CPanel);

JPanel MonitorePanel = new JPanel();
MemoryMonitor MemMonitor = new MemoryMonitor();

```











```

        chkBox.setLayout(new BoxLayout(chkBox, BoxLayout.X_AXIS));
JCheckBox forgetPws= new JCheckBox(" Forget Password ");
        chkBox.add(forgetPws);
ImageIcon iconNew = new ImageIcon("image/ToolBar/lock.gif");
String message = " Only the administrator can enter the Servser,";
String message1 =" then if you administrator, please! entere";
String message2 =" your ID and password : ";
final JOptionPane optionPane =new JOptionPane();
int result = optionPane.showOptionDialog(f2,new java.lang.Object[] {
message,message1,message2,user,IDField,psw,passField,chkBox},

"Lock",JOptionPane.OK_CANCEL_OPTION,

                                JOptionPane.QUESTION_MESSAGE,
                                iconNew, null, null);

String IDNo = new String(IDField.getText());
PSW = new String(passField.getPassword());

IDField.setText("");
passField.setText("");

if (result ==JOptionPane.OK_OPTION ){
    if (forgetPws.isSelected()){
        if (IDNo.equals("")) {
            JOptionPane.showMessageDialog(f2,"Please! fill the ID No.
field !! \n Press Ok to return");
            lockServer();
        }
        else {
            if(!BankRef.hasID(IDNo)){
                JOptionPane.showMessageDialog(f2, "Your ID can not be
found !! \n Press OK to exit.");

                System.exit(0);
            }
            else if(BankRef.hasID(IDNo)){
                BankRef.printClientMsg("\n--Admin-->> The
Administrator Whose ID number is ( "+ IDNo +" ) has forgotten his
password !!!");
                JOptionPane.showMessageDialog(f2, "Your password
will send to your E-mail !! \n Press OK to exit.");
                System.exit(0);
            }
        }
    }
    else if(PSW.equals("") || IDNo.equals("")) {
        JOptionPane.showMessageDialog(f2,"Please! fill all the fields to
log in \n Press Ok to return");
        lockServer();
    }
    else {
        if(BankRef.checkempOk(IDNo, PSW)&&
(!BankRef.IsIDLocked(IDNo))){
            if (BankRef.CheckJOB(IDNo)){
                BankRef.printClientMsg("\n--Admin-->> The
Administratore Whose ID number is ( "+ IDNo +" ) entered successfully
....");
            }else{

```



